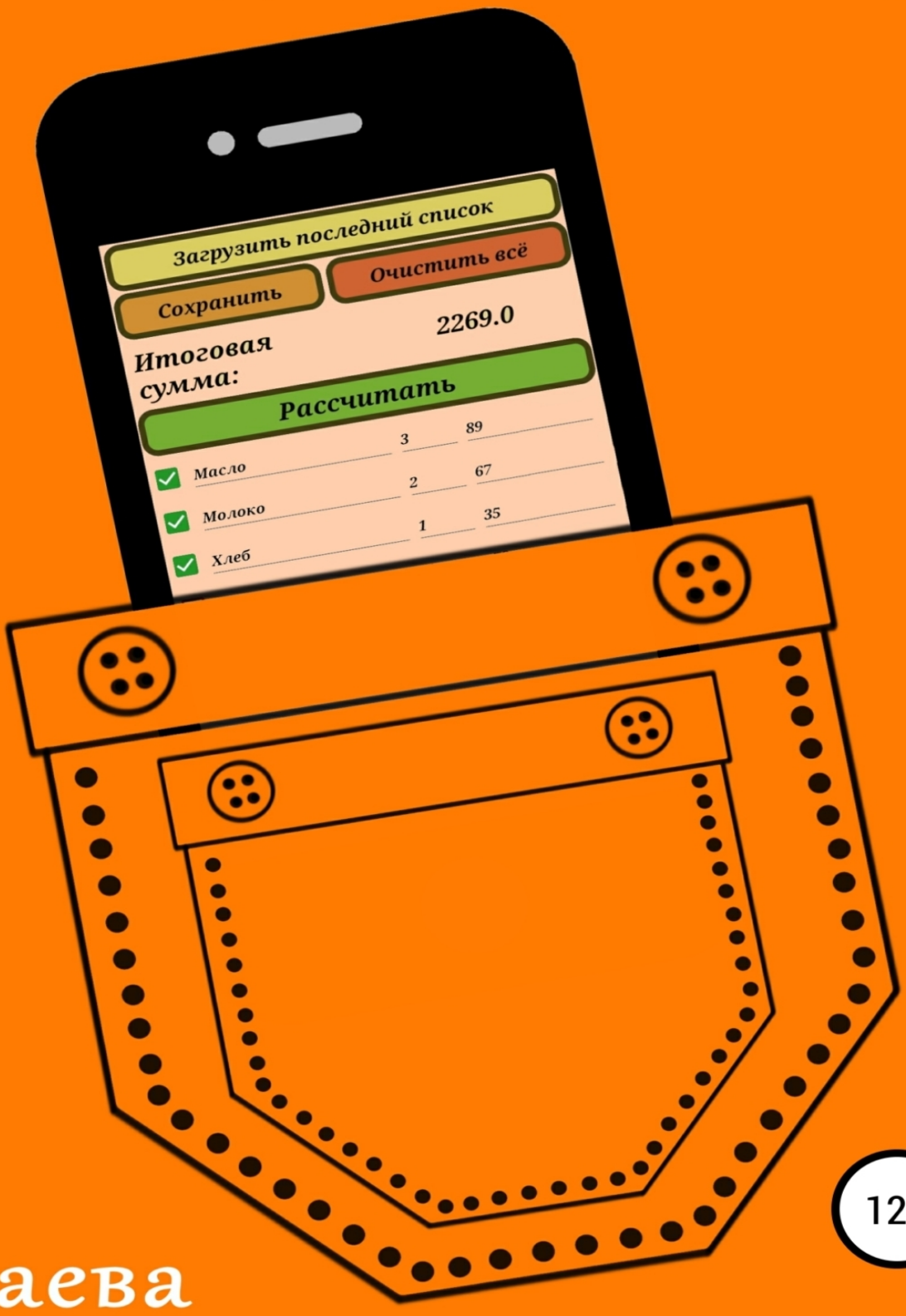
Android Studio

полезные шпаргалки для начинающих



12+

Т. Кучаева

Татьяна Кучаева
Android Studio. Полезные шпаргалки для начинающих

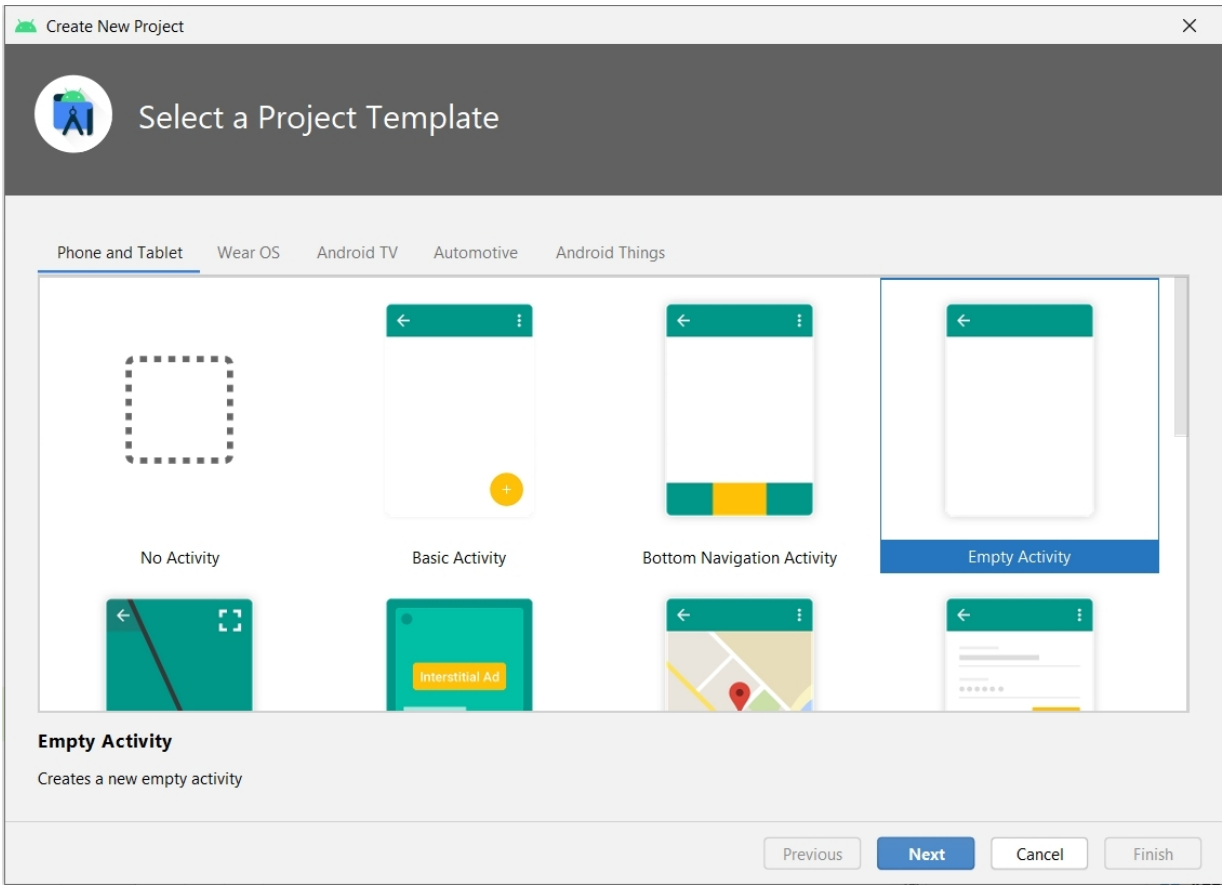
Введение.

Программирование и разработка мобильных приложений на Android уже давно покорила весь мир. С каждым годом всё больше людей хотят научиться создавать приложения и игры на планшет или смартфон. В этой книге есть всё необходимое для того, чтобы научиться самостоятельно разрабатывать приложения, как самые элементарные, так и более функциональные.

Скорее всего Вам уже пришла в голову гениальная идея и Вы хотите воплотить ее в жизнь? Тогда Вы попали по адресу. Вы научитесь правильно создавать приложения, проектировать интерфейс, применять различные способы и методы на практике.

Если Вы читаете эту книгу, предполагается, что Вы уже изучили первые шаги, имеете базовые знания Java, установили на свой компьютер Android Studio и, возможно, создали свой первый проект. Но, скорее всего, Вам не хватает практики, чтобы продвинуться дальше. Именно в этой книге собраны самые необходимые шпаргалки для их практического применения. Все шаги будут обязательно проиллюстрированы и прописаны простым и понятным языком, а также включать в себя фрагменты кода.

Для того, чтобы начать разработку мобильного приложения, необходимо запустить Android Studio и создать новый проект. Для этого на панели управления нажимаем File → New → New Project. Для начала выбираем Empty Activity, что означает «пустая активность», для базового изучения этого будет достаточно. После приобретения достаточного количества знаний можно будет пользоваться шаблонами, которые предоставляет нам среда разработки. Нажимаем Next.

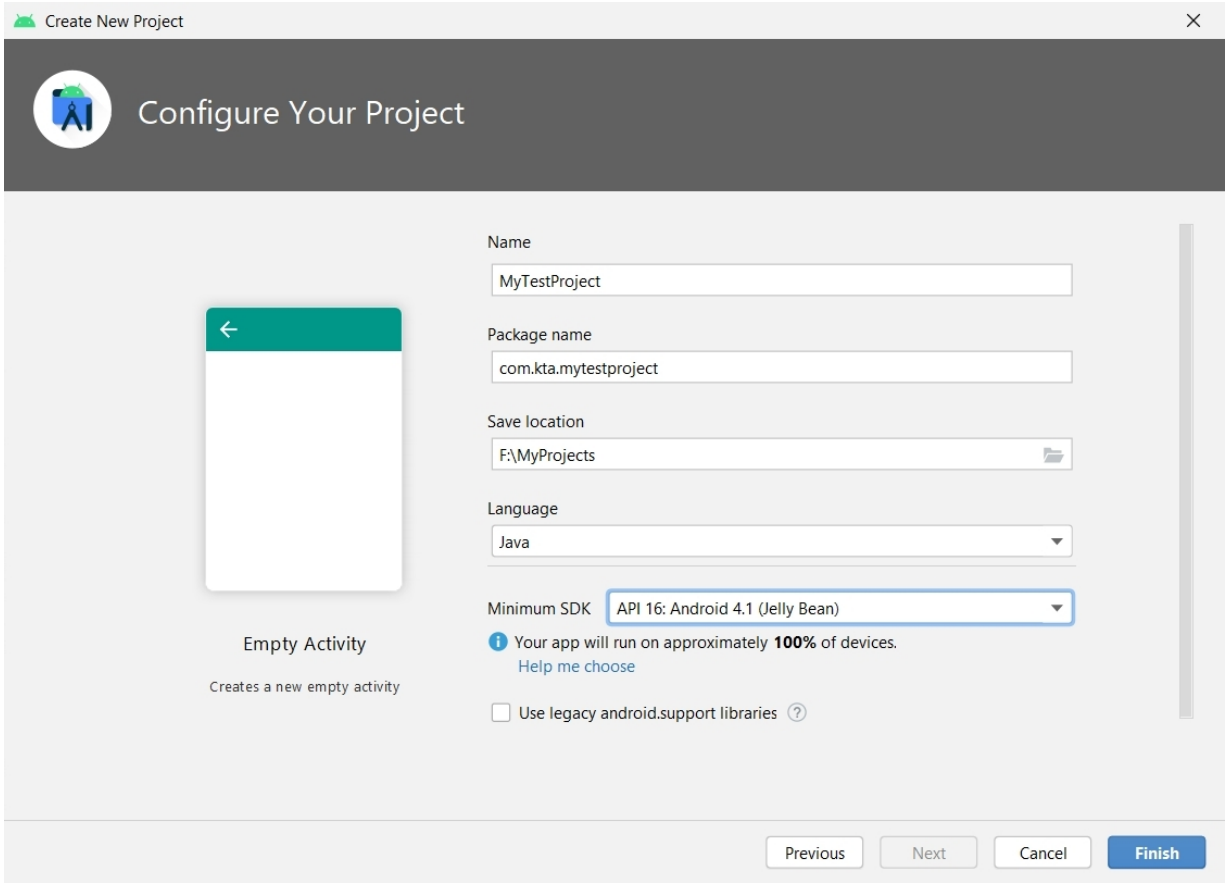


В поле Name необходимо указать имя Вашего проекта, например MyTestProject, или Вы можете задать ему любое другое имя, какое захотите.

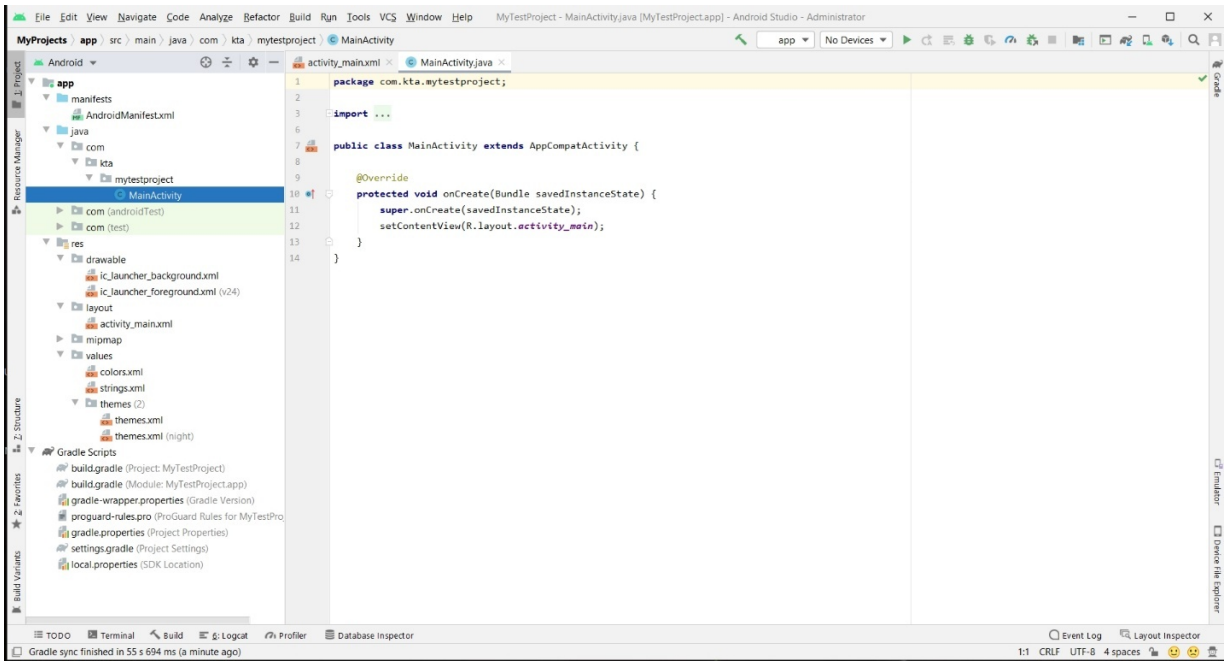
В поле Package name прописываем имя пакета, то есть уникальное название готового приложения. Как правило оно совпадает с именем приложения.

В поле Save location выбираем место на компьютере, куда сохраним проект.

Язык программирования (Language) выбираем Java, а Minimum SDK выбираем такой, чтобы наше приложение поддерживалось на 100% устройств.



Нажимаем Finish и ждём пока Android Studio загрузит наше приложение.



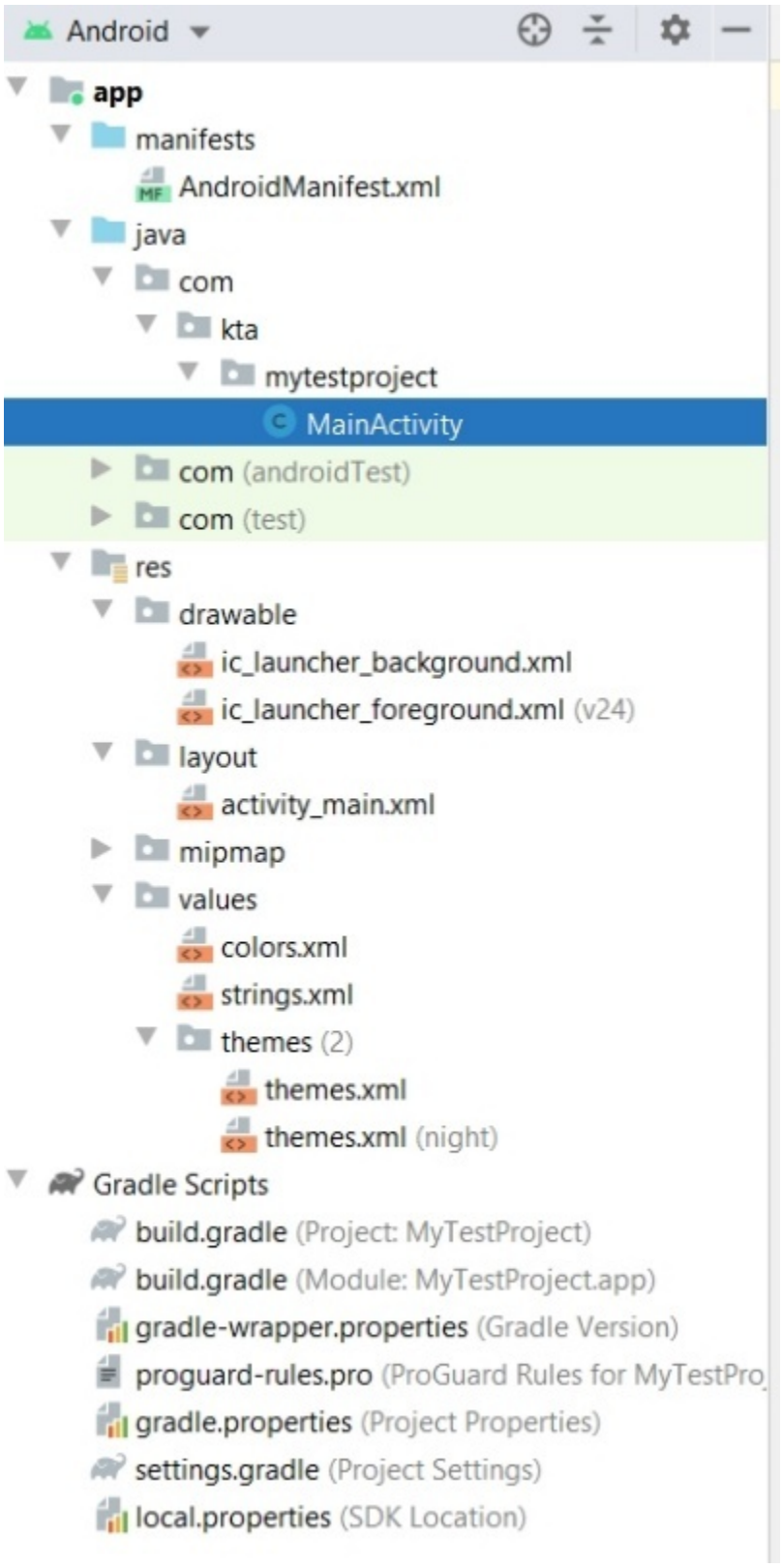
Слева мы видим содержимое нашего проекта, всё, с чем мы будем работать:

AndroidManifest – это некий набор правил, по которым работает готовое приложение. Файл манифеста всегда находится в корневой папке, называется AndroidManifest.xml и содержит в себе всю важную информацию, без которой система просто не сможет запустить приложение.

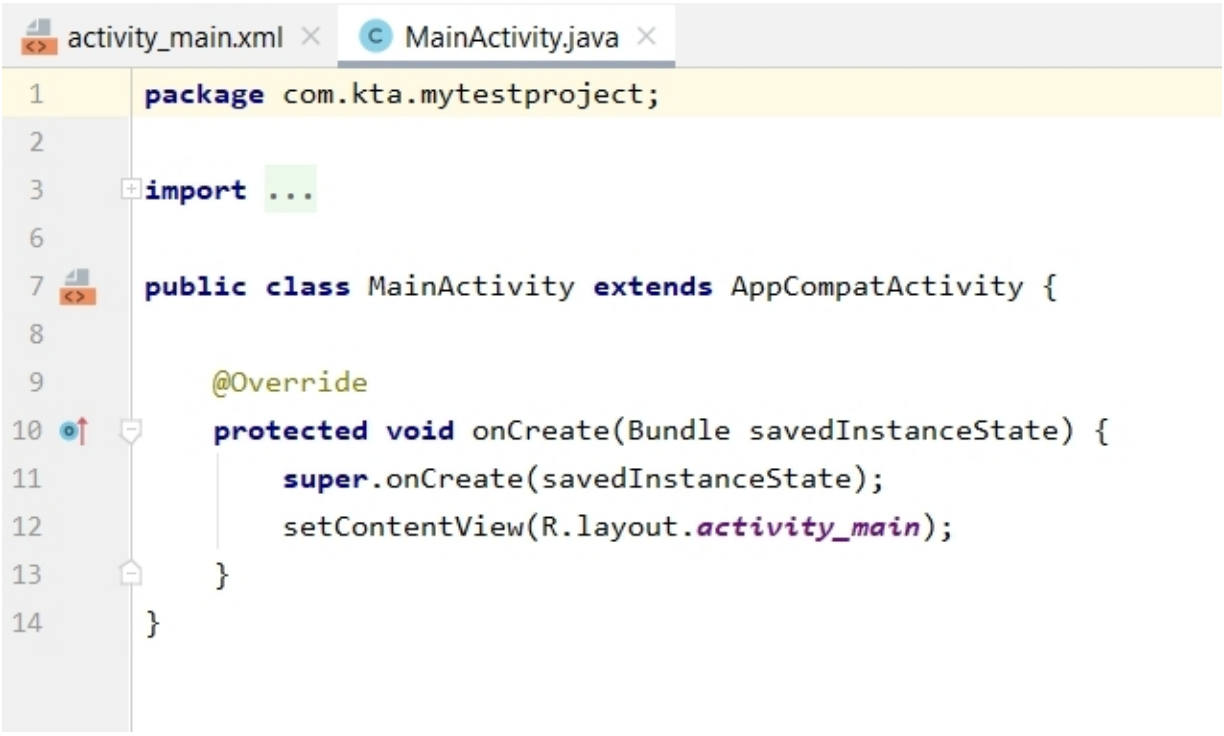
Java-файл, в котором прописывается весь код для взаимодействия приложения с пользователем. Таких файлов может быть сколько угодно много, но главный всегда будет один – MainActivity, именно с него начинается запуск приложения.

res – это папка, которая содержит в себе все необходимые ресурсы для работы приложения: layout, картинки, звуки, музыку, видео, цвета, текст, темы.

Gradle Scripts – система автоматической сборки приложения.


Android
app
manifests
AndroidManifest.xml
java
com
kta
mytestproject
MainActivity
com (androidTest)
com (test)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
activity_main.xml
mipmap
values
colors.xml
strings.xml
themes (2)
themes.xml
themes.xml (night)
Gradle Scripts
build.gradle (Project: MyTestProject)
build.gradle (Module: MyTestProject.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for MyTestPro
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Справа находится поле, где прописывается код для определенных действий и видна вся разметка и структура приложения.

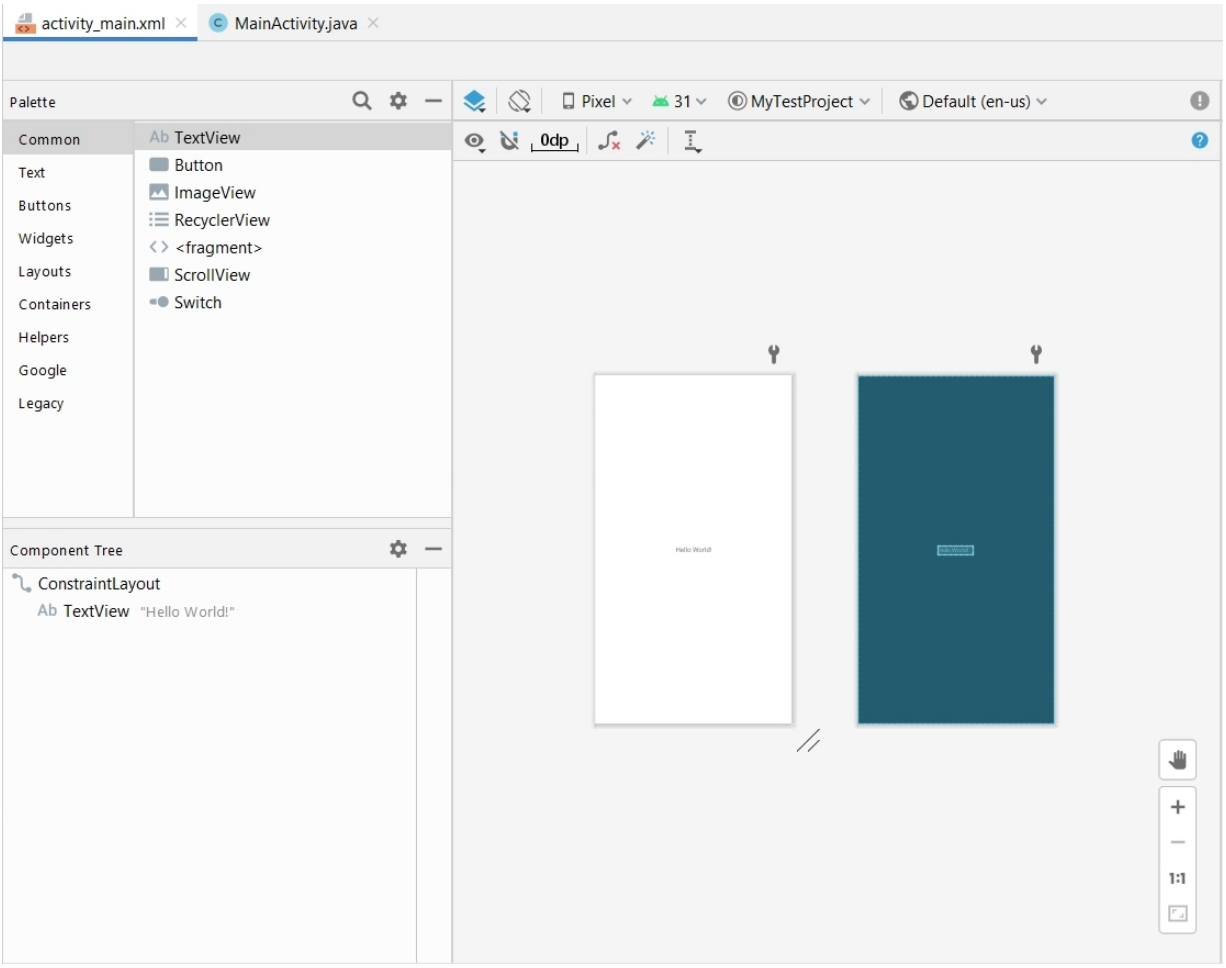


Итак, приложение создано. Но, если мы запустим его на устройстве, не важно – виртуальном или физическом, мы ничего не увидим, кроме белого чистого листа. Это потому, что мы ещё ничего не создали и не добавили в приложение.

Именно этим мы и займёмся в практической части книги.

Шпаргалка № 1. Layout.

Layout – это класс, управляющий как его компоненты будут располагаться и выглядеть на экране телефона в готовом приложении. Каждому Java-файлу соответствует свой xml-файл. Здесь всё просто – в xml находится разметка, мы видим, как располагаются текстовые поля, кнопки и прочее, в java – код, который позволяет всеми этими компонентами управлять. Например, по умолчанию файлу MainActivity.java соответствует файл activity_main.xml

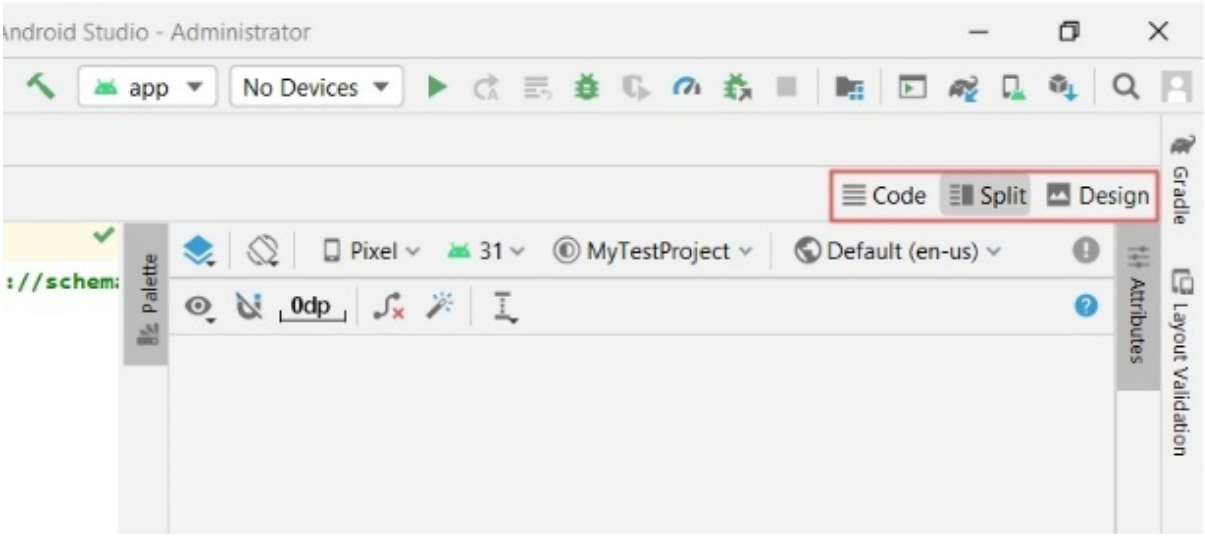


В верхнем правом углу можно заметить три кнопки различного представления:

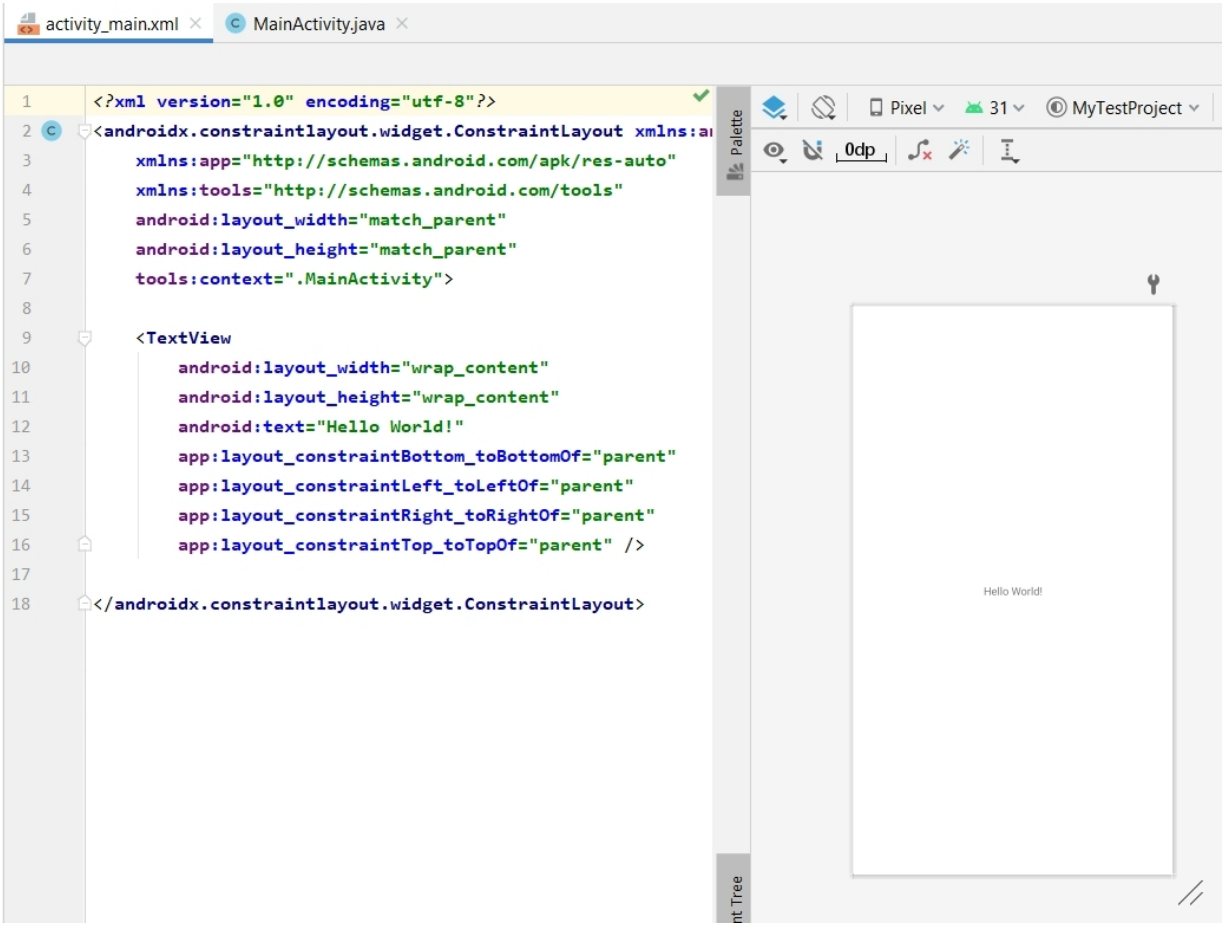
1) Code – означает, что на экране можно видеть только xml разметку, то есть код, который описывает вид и расположение полей, кнопок и прочего.

2) Split – можно видеть как код, так и визуальное представление.

3) Design – видим только визуальное расположение элементов, при этом можем управлять ими в полуавтоматическом режиме. На самом деле, это не очень удобно, поэтому советуем пользоваться вариантом Split.



Именно в таком положении мы можем спокойно менять параметры визуализации, и одновременно видеть все изменения на экране.



Шпаргалка № 2. Макет.

В каждом приложении обязательно присутствуют различные компоненты, такие как текстовые поля, кнопки, чек-боксы, картинки, звуковые дорожки и много другое. Все эти элементы должны каким-то образом располагаться на экране телефона, чтобы пользователь мог с ними взаимодействовать. И именно за это отвечает макет (или разметка). Если представить себе, что макет – это контейнер, а элементы – например кубики, то говоря простыми словами: мы раскладываем кубики в контейнер так, как нам хочется, как того требует ситуация.

Разметки бывают нескольких видов, некоторые из которых мы рассмотрим отдельно.

1) FrameLayout – это, пожалуй, самый простой вид разметки. Как правило, это пустое пространство на экране, которое возможно заполнить элементами только путём прикрепления их к верхнему левому углу экрана. В такой разметке никак нельзя определить желаемое местоположение для объектов. Каждый последующий добавленный объект будет накладываться поверх предыдущих, при этом частично или полностью затеняя их.

2) LinearLayout – название говорит само за себя: все элементы в данной разметке располагаются в одну линию. Макет автоматически выравнивает все находящиеся в нём объекты в одном заданном направлении – либо вертикально, либо горизонтально. Все элементы становятся один за другим в ряд и имеют одинаковый размер, за исключением случаев, когда для отдельных элементов прописывается конкретный размер или конкретный вес. Об этом мы поговорим чуть позже и рассматривать будем уже на практике.

3) TableLayout – табличная разметка, которая позиционирует добавленные элементы в строки и столбцы, она не отображает линии для них, и вполне может иметь строки с разным количеством ячеек. При формировании такой разметки некоторые из ячеек при необходимости можно оставлять пустыми либо заполнять другими макетами.

4) RelativeLayout – это относительная разметка, в которой все элементы расположены так, что, если первый элемент расположен по

центру экрана, другие элементы, будут выравниваться относительно первого элемента.

5) ConstraintLayout – представляет собой контейнер, который позволяет создавать гибкие интерфейсы. Для определения позиции элемента внутри данного макета необходимо указать ограничения либо по горизонтали, либо по вертикали. Позиционирование может производиться как относительно границ самого макета, так и относительно любого другого элемента внутри разметки.

6) ScrollView – скроллинг. Используется при больших объёмах текста, картинок и прочего, которые не помещаются на экран и требуется прокрутка. ScrollView внутри себя может иметь только один дочерний элемент, например LinearLayout, а в этом макете уже размещаются остальные элементы.

Шпаргалка № 3. Параметры.

Абсолютно все макеты и элементы, которые содержит в себе приложение, должны иметь некие параметры: название, размер (высота и ширина), вес, ориентация, задний фон, цвет и многое другое. В этой главе мы рассмотрим основные, самые важные параметры для разметок и их элементов.

android:id="@+id/abc" – ID элемента (имя), пожалуй самый важный параметр, так как при любом действии Android Studio будет обращаться к элементу именно по ID.

android:layout_width="match_parent" – ширина элемента, бывает двух типов **"match_parent"** – будет растянут на весь контейнер, или **"wrap_content"** – займёт места столько, сколько внутри будет контента (например, текста). Также, при необходимости, можно указывать размер вручную, для этого необходимо указать следующий параметр в пикселях, например **"200dp"**. Цифра может быть любая, хоть 121, хоть 367.

android:layout_height="wrap_content" – высота элемента. Все параметры такие же, как и у ширины.

android:layout_weight="1" – вес. Данный параметр чаще всего используется в разметке LinearLayout, когда элементы должны занимать на экране определенное количество места, но делать это должны автоматически, а не вручную. Параметр может быть любой, начиная от числа 0,1.

android:orientation="vertical" – ориентация макета. Бывает **"vertical"** – вертикальная, и **"horizontal"** – горизонтальная.

android:background="@color/cherry" – задний фон. На любой макет или элемент можно установить задний фон. Это может быть просто сплошной цвет, который взят из базы цветов в разделе color (**"@color/cherry"**), или прописан кодом (**"#D12B68"**), а также это может быть любая картинка (**"@drawable/print"**, **"@mipmap/print"**).

android:gravity="center" – гравитация. Определяет местоположение текста внутри элемента, в данном примере – по центру.

android:layout_gravity="center_horizontal" – определяет местоположение элемента внутри контейнера. Так же, как и **android:gravity** может быть разных видов: **"top"** – сверху, **"bottom"** – снизу, **"left"** – слева, **"right"** – справа.

android:text="@string/text" – данный параметр используется чаще всего для текстовых элементов или кнопок, которые содержат в себе надписи, подписи, тексты.

android:paddingTop="10dp" – параметр, который используется для отступа содержимого элемента внутри этого самого элемента сверху, значение может быть любое.

android:paddingBottom="10dp" – отступ снизу.

android:paddingLeft="10dp" – отступ слева.

android:paddingRight="10dp" – отступ справа.

android:padding="20dp" – используется для равномерного отступа со всех сторон.

android:layout_marginTop="15dp" – в отличие от предыдущего, данный параметр используется для отступа элемента внутри контейнера, либо для отступа от соседнего элемента, так же может быть сверху (**layout_marginTop**), снизу (**layout_marginBottom**), слева (**layout_marginLeft**) и справа (**layout_marginRight**). Значение в dp может быть любое.

android:layout_margin="25dp" – используется для равномерного отступа со всех сторон.

style="@style/MyTextStyle" – если приложение содержит в себе множество элементов с одинаковыми параметрами, гораздо удобнее создать стиль, указав его для элемента. Так и код смотрится более читабельно, и, если будет необходимость что-то в этих параметрах изменить – изменять придётся только в одном месте, а не в 10, 20, 50 и более элементах.

Конечно, это лишь малая часть из всех существующих параметров, но это самые наиболее часто используемые и нужные. Более подробно мы ещё будем их рассматривать на практике в последующих главах.

Шпаргалка № 4. Текстовое поле.

Ни одно приложение не обходится без текста, каких-либо надписей, подписей, информации о приложении, в некоторых случаях даже ввода текста самим пользователем. За всё это отвечает текстовое поле. Об этом и поговорим.

Текстовые поля в редакторе бывают двух видов TextView и EditText.

TextView – элемент, содержащий в себе определённое количество любого текста и предназначен он только для его просмотра, без возможности редактирования пользователем.

EditText – это также текстовое поле, но уже с возможностью ввода или редактирования текста.

Добавляем текстовое поле в приложение:

<**TextView**
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>

По умолчанию только что созданное текстовое поле всегда имеет только два параметра, ширину и высоту.

В первую очередь нужно задать имя, для этого добавляем строку:

android:id="@+id/tv1"

Назовем его, например, tv1. Но в Вашем случае название может быть каким угодно, главное, чтобы оно было, и Вы могли к нему обратиться в случае необходимости.

Определяемся с текстом, допустим, это будет что-то приветственное, «Добро пожаловать!»

В ресурсах ищем файл strings.xml, там хранятся все строковые ресурсы, используемые в приложении, создаём строку:

<**string name="str1"**>Добро пожаловать!</**string**>

Добавляем текст в наше поле:
android:text="@string/str1"

Чтобы текст на экране мобильного телефона выглядел красиво и приятно читался, необходимо прописать параметры, которые его улучшат, так как Android Studio по умолчанию предлагает стандартный вид.

Задаём размер текста:
android:textSize="35sp" – Цифра может быть любой, начиная с единицы.

А чтобы текст не был БОЛЬШИМИ БУКВАМИ, добавим следующий параметр:
android:textAllCaps="false"

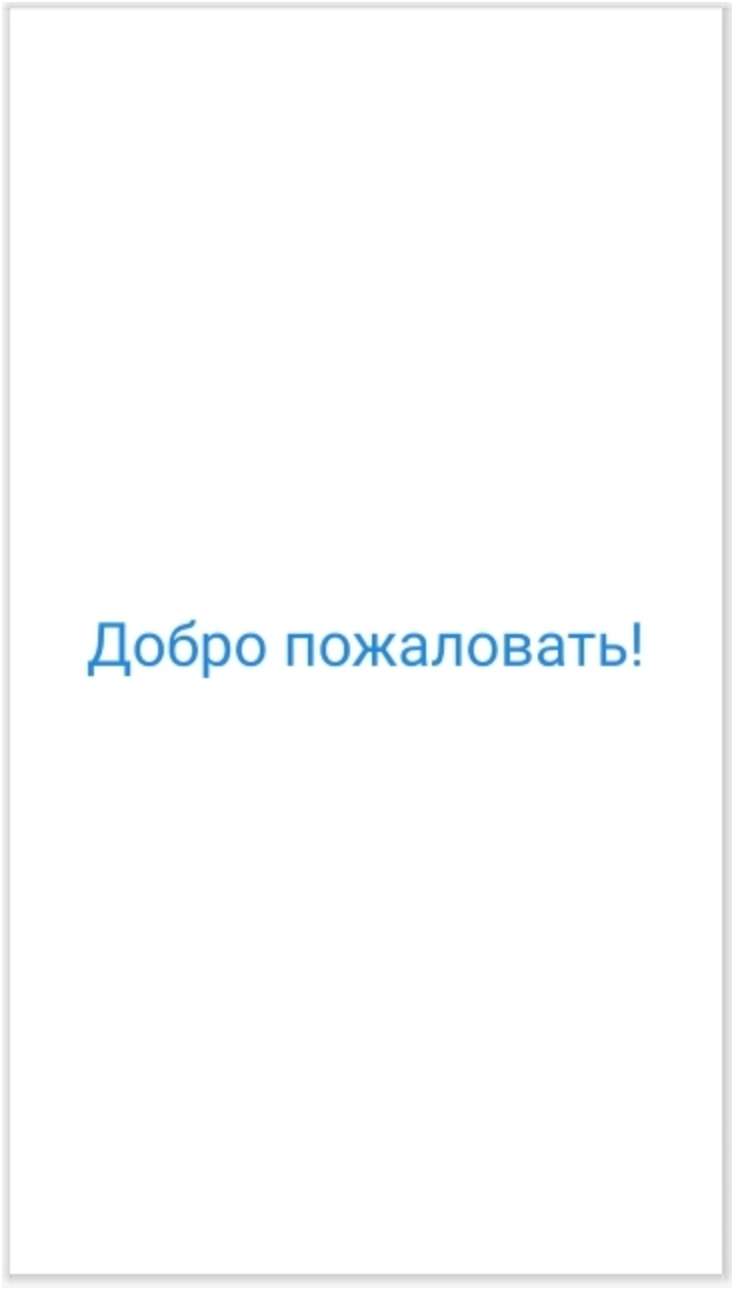
Можно поменять цвет текста. Допустим, мы хотим сделать его голубым. Для этого переходим в ресурсы, открываем вкладку colors.xml и добавляем строку:
<**color name="blue"**>#1E88E5</**color**>

Возвращаемся к нашему текстовому полю и добавляем строку:
android:textColor="@color/blue"

Теперь поместим наше текстовое поле на середину экрана, для этого добавим строку в параметры макета:
android:gravity="center"

И строку в параметры элемента:

android:gravity="center"
android:layout_gravity="center"



Таким образом мы получили приветственный экран нашего приложения. Пользователь увидит его, как только приложение

загрузится. Вот его полный код:

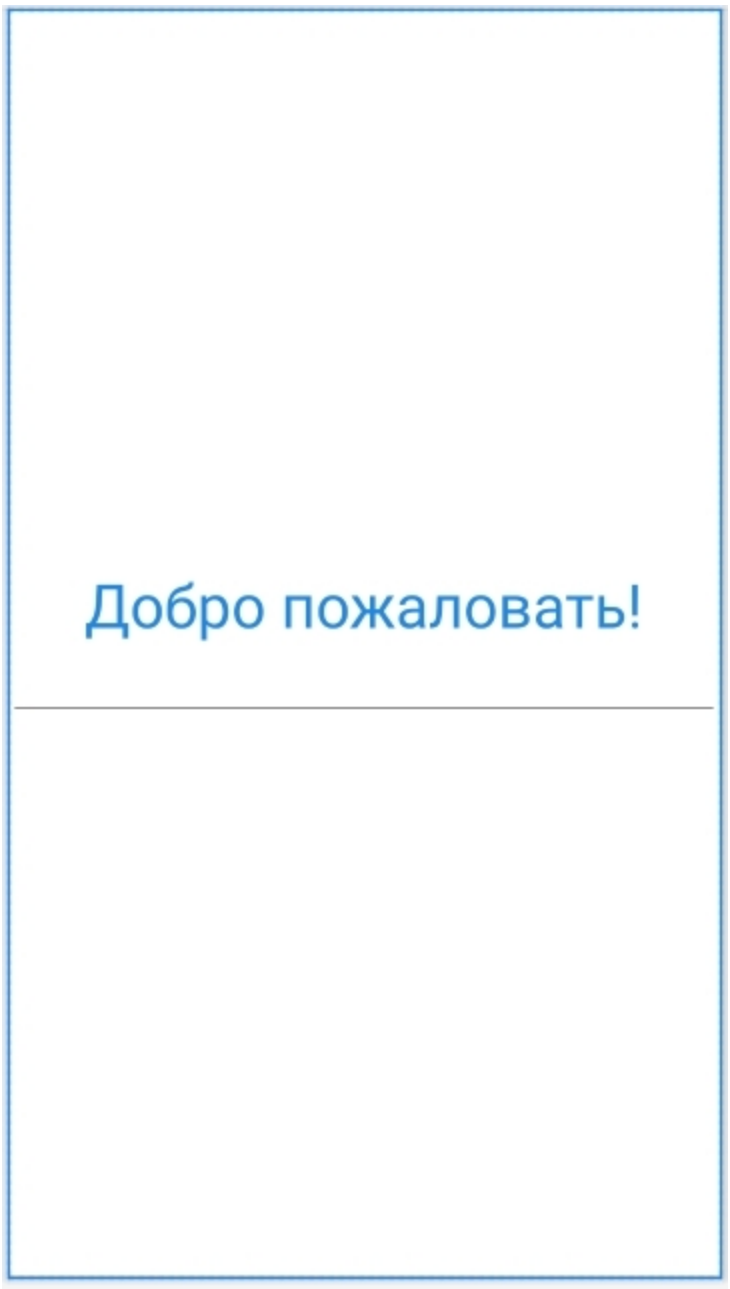
```
<TextView
android:id="@+id/tv1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/str1"
android:textColor="@color/blue"
android:textSize="35sp"
android:textAllCaps="false"
android:gravity="center"
android:layout_gravity="center"/>
```

Но, давайте рассмотрим ещё вариант текстового поля EditText, в которое можно что-то вписывать, например имя.

Добавим такое поле чуть ниже.

```
<EditText
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
```

Ширину установим "match_parent" – то есть на всю ширину макета, а высоту оставим по количеству контента. Теперь в это поле пользователь может ввести своё имя.



Но, как же он узнает, что туда нужно вводить, если мы ему об этом не подскажем? Да и вообще, поле выглядит как обычная полоса, совсем не похоже, что это текстовое поле. И оно как бы «прилипает» к первому приветственному полю, что совсем не красиво смотрится. Давайте это исправим. Для начала сделаем отступы:

android:layout_margin="40dp"

А затем сделаем для пользователя подсказку. В строковый ресурс добавим:

<**string name="str2"**>Введите своё имя</**string**>

А в текстовое поле :

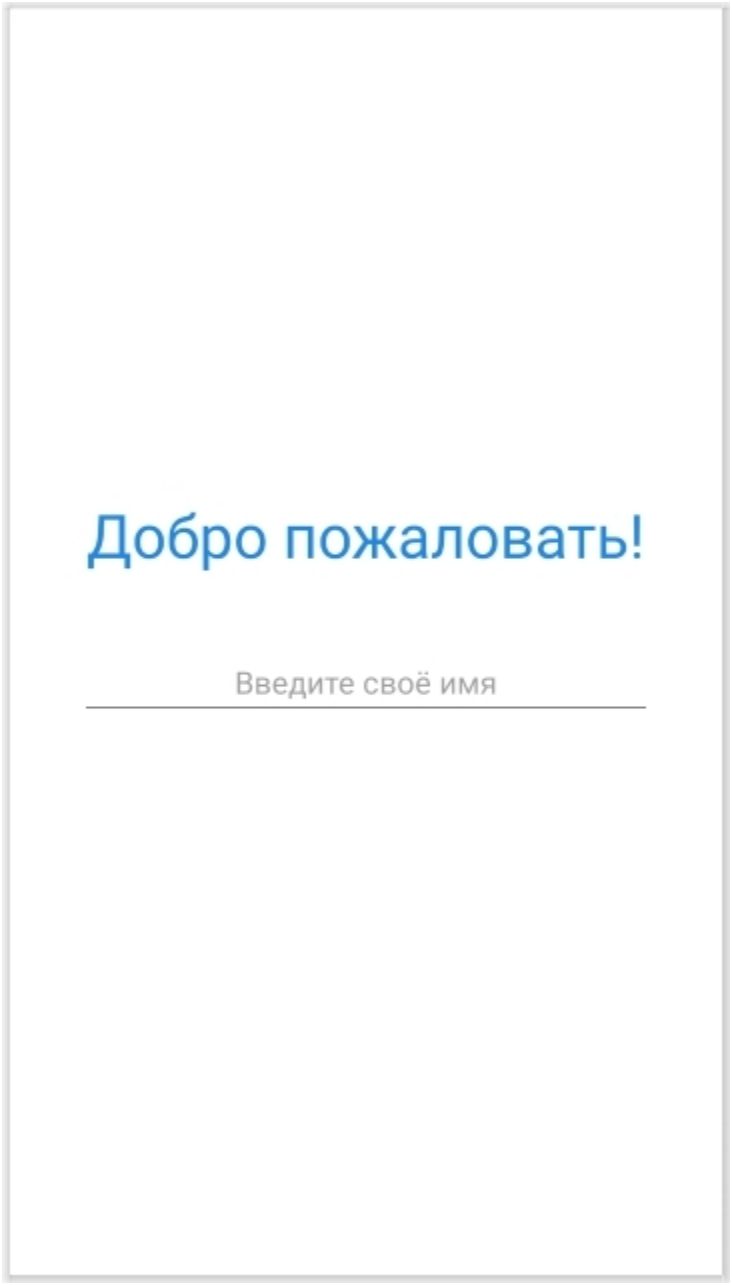
android:hint="@string/str2"

И выравнивание текста посередине:

android:gravity="center"

Итого, мы получили:

<**EditText**
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="40dp"
android:hint="@string/str2"
android:gravity="center"/>



Теперь пользователь может ввести своё имя. Подобным образом можно делать поля для ввода текста, цифр, номера телефона, логина, пароля и прочее.

Шпаргалка № 5. Красивый текст.

Обычный текст в текстовом поле – это ещё не всё. К примеру, мы хотим задать задний фон, скажем – зелёный. Добавляем цвет в файл

colors.xml:

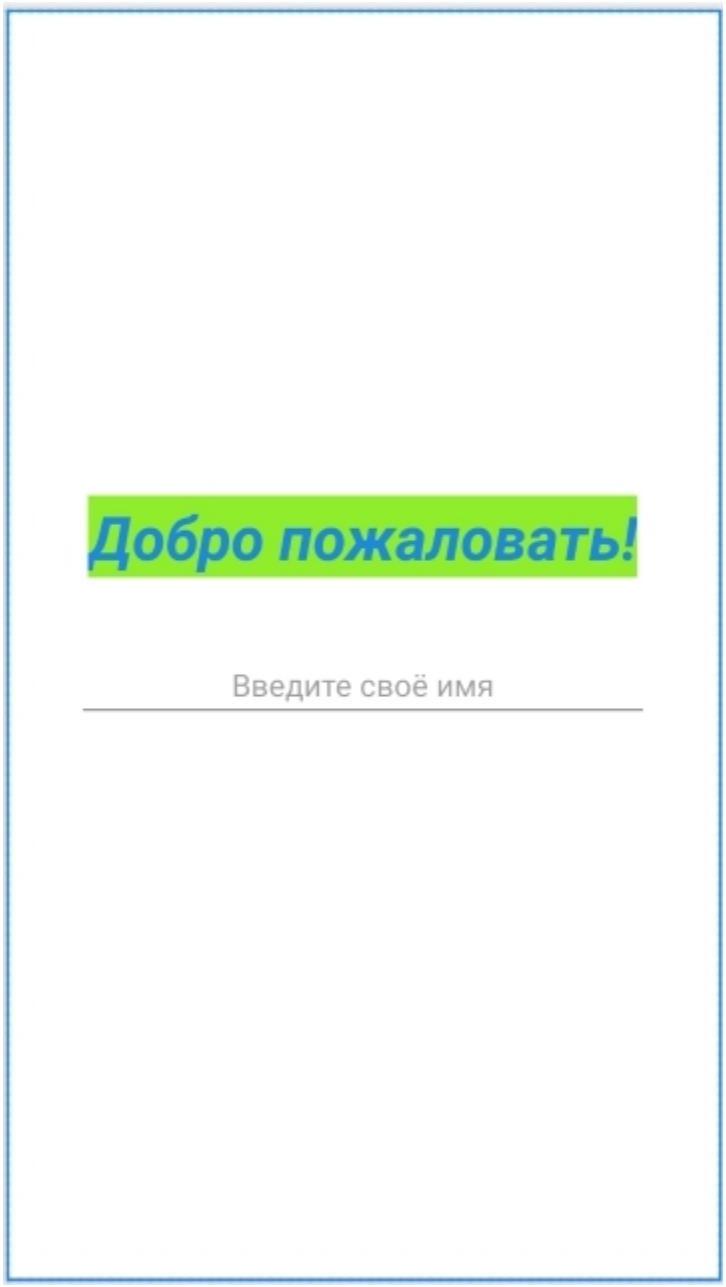
<**color name="green"**>#8FED2C</**color**>

Устанавливаем цвет заднего фона на текстовое поле строкой:

android:background="@color/green"

А сам текст мы хотим сделать жирным и курсивом, тогда нам нужно это указать:

android:textStyle="bold|italic"



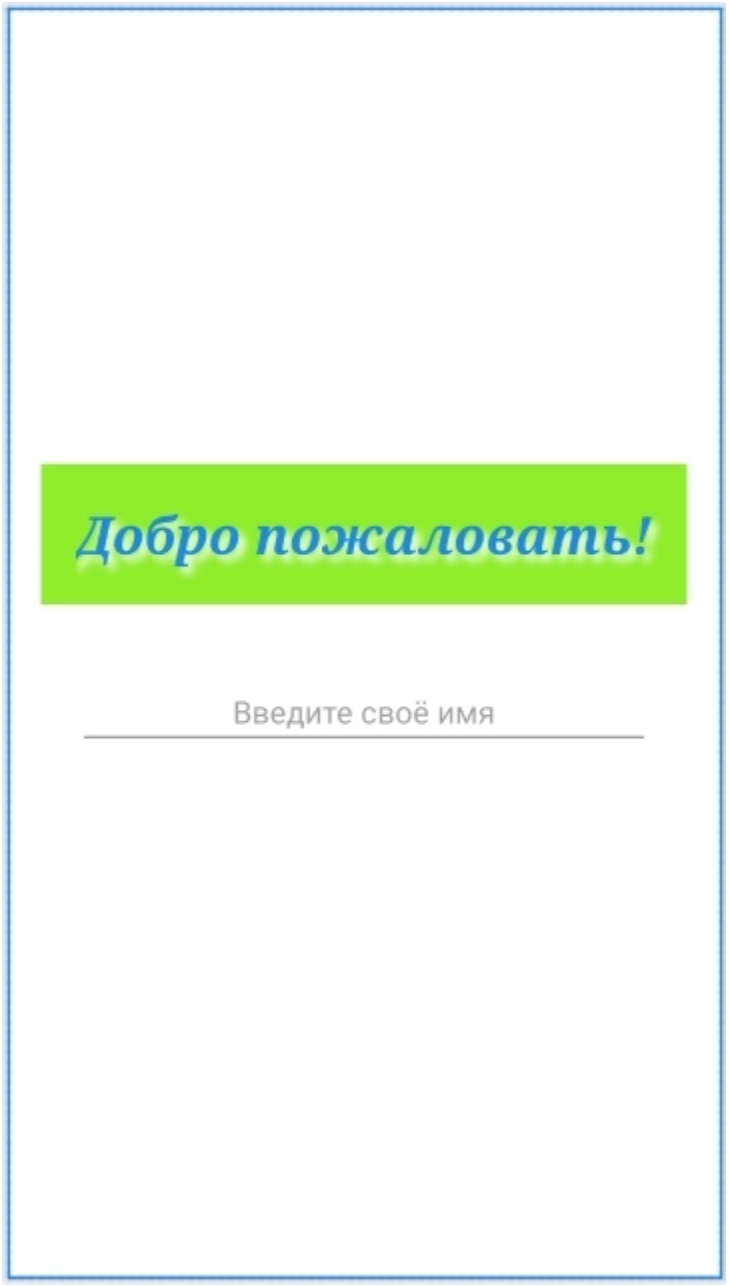
Вроде всё хорошо, но надпись как будто прилипла к стенкам и смотрится не очень красиво. Здесь на помощь придут отступы, задать которые можно следующим способом:

android:padding="20dp"

Также можно изменить шрифт, чтобы буквы были более красивыми:

android:typeface="serif"

И добавить тексту тень:
android:shadowColor="@color/white"
android:shadowDx="10"
android:shadowDy="10"
android:shadowRadius="10"



Если после изменений надпись не помещается в одну строку, можно подкорректировать размер, например:

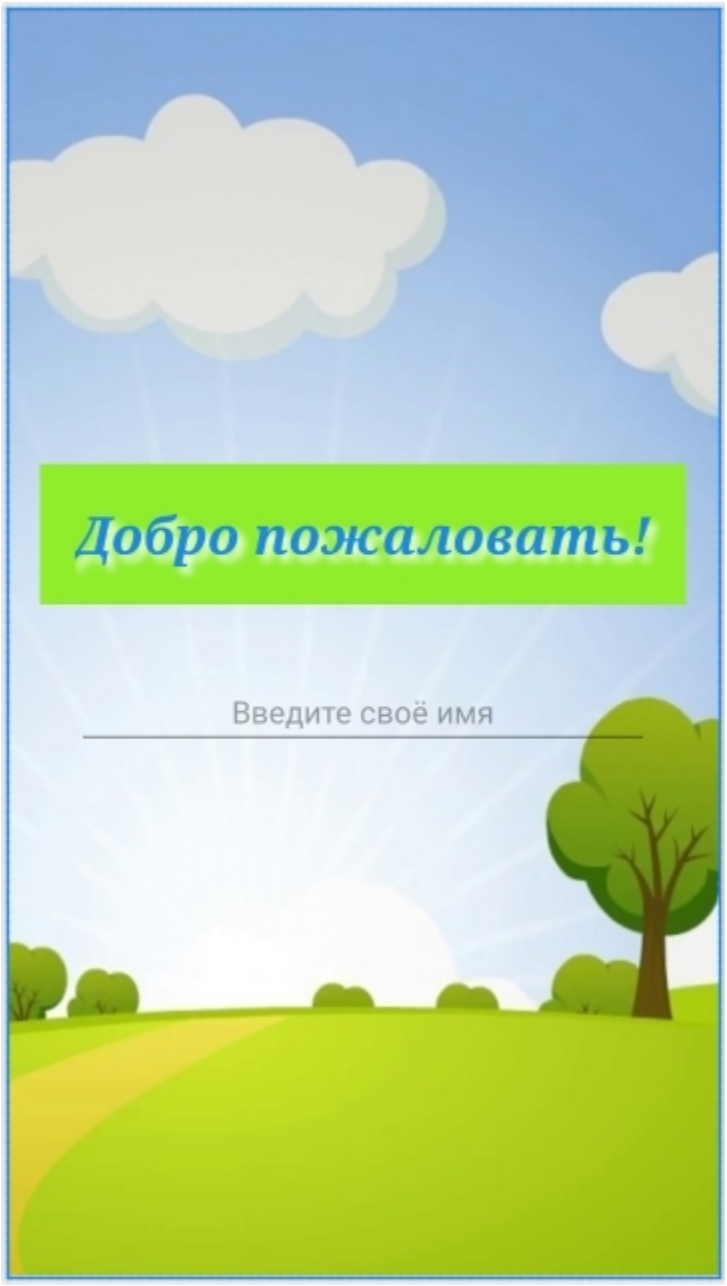
android:textSize="30sp"

Шпаргалка № 6. Задний фон.

Вот теперь всё отлично! Только на заднем фоне хочется видеть не белый экран, а например, картинку. Для этого нужно установить

background для макета. Картинку можно нарисовать самому, если у Вас есть такая возможность и навыки художника, либо скачать из интернета. Назовём файл «fon». Затем скопируем его и вставим в папку drawable. А для макета укажем его в качестве заднего фона:

android:background="@drawable/fon"



Такой вот незамысловатый получился главный экран приложения. А вот полный код, который получился у нас в файле activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:orientation="vertical"
android:background="@drawable/fon"
tools:context=".MainActivity">


<TextView
android:id="@+id/tv1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/str1"
android:textColor="@color/blue"
android:textSize="30sp"
android:textAllCaps="false"
android:gravity="center"
android:layout_gravity="center"
android:background="@color/green"
android:textStyle="bold|italic"
android:padding="20dp"
android:typeface="serif"
android:shadowColor="@color/white"
android:shadowDx="10"
android:shadowDy="10"
android:shadowRadius="10"/>


<EditText
```

android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="40dp"
android:hint="@string/str2"
android:gravity="center"/>

</**LinearLayout**>

Шпаргалка № 7. Стили.

Как видите, код получился объемным, и это мы ещё даже не старались. Только один объект TextView занял столько строк. А что, если таких элементов будет 10, 20, 50? Как сделать так, чтобы написать большой код один раз, а в параметрах указать всё это одной строкой? Для этого существуют стили, или темы. Находятся они в ресурсах, в файле themes.xml, там можно создавать и хранить сколько угодно тем, главное – каждую называть уникальным именем, чтобы в дальнейшем её легко было применить. Переходим в файл и создаем нужную нам тему, назовём её text_style1, так мы будем понимать, что стиль у нас создан для текста и он под номером 1, так как стилей будет много, будут меняться цвета, размеры шрифта, цвет заднего фона и т.д.:

<**style name="text_style1"**>
<**item name="android:layout_width"**>wrap_content</**item**>
<**item name="android:layout_height"**>wrap_content</**item**>
<**item name="android:textColor"**>@color/blue</**item**>
<**item name="android:textSize"**>30sp</**item**>
<**item name="android:textAllCaps"**>false</**item**>
<**item name="android:gravity"**>center</**item**>
<**item name="android:layout_gravity"**>center</**item**>
<**item name="android:background"**>@color/green</**item**>
<**item name="android:textStyle"**>bold|italic</**item**>
<**item name="android:padding"**>20dp</**item**>
<**item name="android:typeface"**>serif</**item**>
<**item name="android:shadowColor"**>@color/white</**item**>
<**item name="android:shadowDx"**>10</**item**>

<item name="android:shadowDy">10</item>
<item name="android:shadowRadius">10</item>
</style>

Теперь стираем в параметрах нашего текстового поля все эти строки, кроме имени и ссылки на текст, а вместо них пишем всего одну:

<TextView
android:id="@+id/tv1"
android:text="@string/str1"
style="@style/text_style1"/>

Таким образом, часть кода у нас сократилась, а вид не изменился. К тому же, если мы применим стиль к 100 таким текстовым полям, а потом вдруг захотим внести изменения, сделать это будет гораздо проще и всего один раз – в файле стилей. Если стиль не применять – изменять придется во всех 100 элементах. Преимущества такого подхода налицо.

Шпаргалка № 8. Кнопка.

Ну какое же приложение обходится без кнопки? Конечно, в любом приложении она есть, и даже ни одна. Если нужно что-то загрузить, сохранить, рассчитать, начать, завершить, подтвердить и многое-многое другое, то используется компонент button, то есть кнопка. Сделаем такую кнопку, которая будет перенаправлять нас на другую активность и якобы начинать какую-то игру или запускать какой-либо процесс. Назовём её «Начать».

Создадим для нашей кнопки стиль, строковый ресурс и цвет заднего фона.

Для файла themes.xml пропишем:

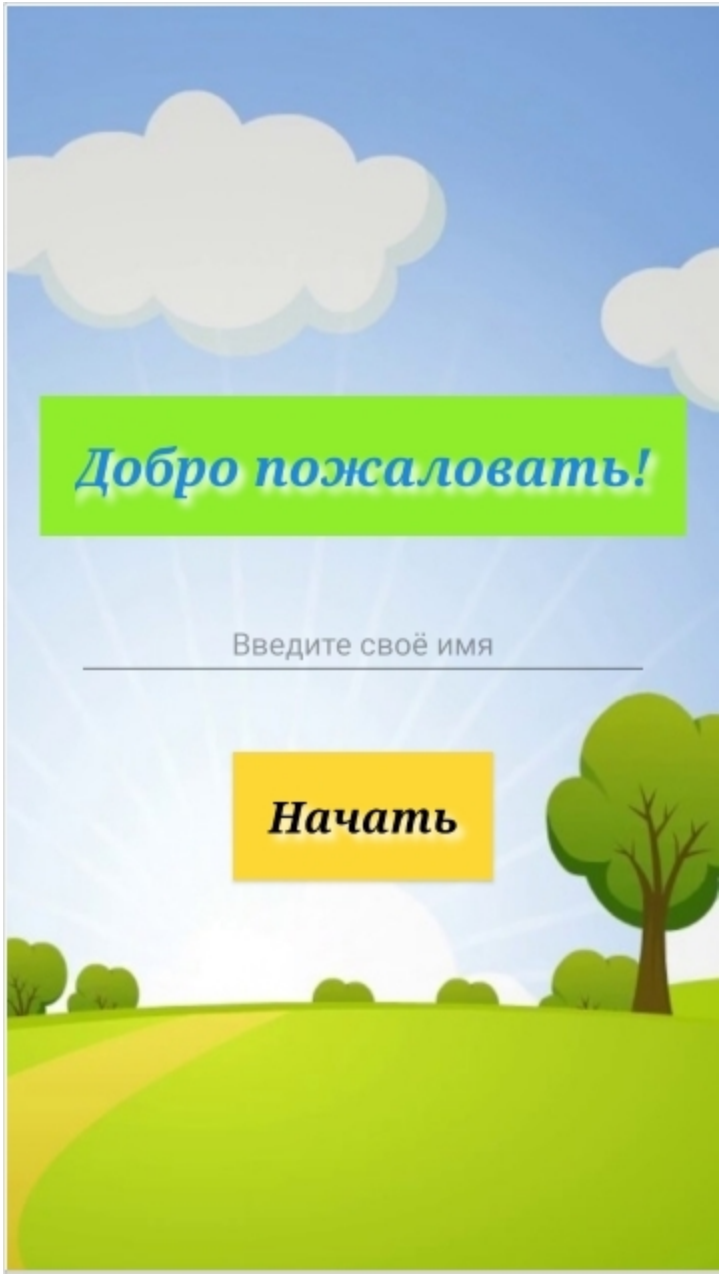
<style name="btn_style1">
<item name="android:layout_width">wrap_content</item>
<item name="android:layout_height">wrap_content</item>
<item name="android:textColor">@color/black</item>

<item name="android:textSize">25sp</item>
<item name="android:textAllCaps">false</item>
<item name="android:gravity">center</item>
<item name="android:layout_gravity">center</item>
<item name="android:background">@color/yellow</item>
<item name="android:textStyle">bold|italic</item>
<item name="android:padding">20dp</item>
<item name="android:typeface">serif</item>
<item name="android:shadowColor">@color/white</item>
<item name="android:shadowDx">10</item>
<item name="android:shadowDy">10</item>
<item name="android:shadowRadius">10</item>
</style>

Для файла strings.xml:
<string name="str3">Начать</string>

А для файла colors.xml:
<color name="yellow">#FDD835</color>

В activity_main.xml добавим код для кнопки, зададим ей параметры:
<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_start"
android:text="@string/str3"
style="@style/btn_style1">
</androidx.appcompat.widget.AppCompatButton>



Шпаргалка № 9. Рабочая кнопка.

Наша кнопка готова. Она есть на экране, имеет название и стиль.

Но она пока не работает, если мы будем на неё нажимать, то ничего не произойдёт. Потому что для этого необходимо написать определённый код.

В данном случае мы хотим, чтобы наша кнопка отправила нас на другую активность, то есть на другую страницу приложения. Она пока

будет пустой, мы заполним её чуть позже.

Для начала эту страницу нужно создать. Кликаем по файлу MainActivity правой кнопкой мыши, нажимаем New → Java Class, назовём нашу вторую активность например Activity2 (Вы можете придумать своё собственное название), а в папке layout создаём новый файл xml, назовём его так же, как и java-файл – activity2, и чтобы их связать между собой, пропишем в java-файле следующий код:

```
package com.kta.mytestproject;


import androidx.appcompat.app.AppCompatActivity;


import android.os.Bundle;


public class Activity2 extends AppCompatActivity {


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity2);
}
}
```

Таким образом, метод setContentView сообщает Android, что при работе необходимо обратиться к активности activity2.xml.

Возвращаемся к нашему MainActivity.java и добавляем код, который укажет наши намерения перейти из одной активности в другую. Для этого используется слушатель OnClickListener.

```
Button start = (Button) findViewById(R.id.btn_start);
```

```
start.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
try {
Intent intent = new Intent(MainActivity.this, Activity2.class);
startActivity(intent);finish();
} catch (Exception e){


}
}
});
```

Теперь разберём всё более детально:

Button start = (Button) findViewById(R.id.btn_start);

При выполнении этой строки кода создается объект класса Button с именем start. Метод findViewById() получает идентификатор компонента, то есть находит его по имени (в данном случае btn_start).

start.setOnClickListener – слушатель связывается с кнопкой.

Далее происходит реализация метода onClick().

Intent intent = **new** Intent(MainActivity.**this**, Activity2.**class**);

Мы говорим системе, что находимся здесь (MainActivity.**this**,), а перейти хотим сюда (Activity2.**class**).

Запускаем нашу активность Activity2 и закрываем за собой MainActivity:

startActivity(intent);finish();

Все активности должны реализовать метод onCreate(), он вызывается при создании объекта активности и используется для настройки основных параметров. Поэтому получиться у Вас должно вот так:

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button start = (Button) findViewById(R.id.btn_start);

        start.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    Intent intent = new Intent( packageContext: MainActivity.this, Activity2.class);
                    startActivity(intent);finish();
                } catch (Exception e){

                }
            }
        });


    }
}
```

Шпаргалка № 10. Новая активность.

Если в данный момент мы нажмём на кнопку «Начать», несмотря на то что код для перехода уже готов, на другую активность приложение всё равно не перейдёт, потому что активность не указана в манифесте.

Это важно! Никогда не забывайте прописывать все новые активности в файле AndroidManifest.xml.

Сделать это можно следующим способом: переходим в файл манифеста и после основной активности добавляем код:

<activity
android:name="com.kta.mytestproject.Activity2"
<intent-filter>
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

Теперь, при нажатии на кнопку старт, мы будем попадать на активность приложения Activity2.

Шпаргалка № 11. Красивая кнопка.

Мы создали кнопку и даже сделали её активной, она работает и совершает переход на другую активность. Но, к сожалению, визуально мы не видим ничего, когда на неё нажимаем.

Чтобы сделать кнопку кликабельной и более красивой чем она есть сейчас, необходимо подредактировать её стиль.

Для начала добавим ещё один цвет в наш файл colors.xml:

<**color name="orange"**>#FF8E00</**color**>

Таким цветом будет окантовка не нажатой кнопки и заливка нажатой, чтобы она визуально отличалась, и пользователь видел, что он кнопку действительно нажал.

В папке drawable создаём три файла xml:

1) style_btn_no_press.xml – для определения стиля не нажатой кнопки;

2) style_btn_press.xml – для определения стиля нажатой кнопки;

3) style_btn.xml – для применения стиля кнопки.

В первом файле style_btn_no_press.xml добавляем следующий код:

<?**xml version="1.0" encoding="utf-8"**?>

<**shape**
xmlns:android="http://schemas.android.com/apk/res/android">

<**stroke**
android:width="5dp"
android:color="@color/orange">
</**stroke**>

<**corners**
android:radius="20dp">
</**corners**>

<**solid**
android:color="@color/yellow">
</**solid**>

</**shape**>

Stroke – означает какой будет окантовка кнопки:
android:width="8dp" – толщина линии;
android:color="@color/orange"> – цвет линии, в данном случае оранжевый.

Corners – определяет скругление углов:
android:radius="30dp"> – значение может быть любым, чем больше, тем круглее.

Solid – заливка кнопки, то есть основной задний фон:
android:color="@color/yellow"> – определяем цвет, оставим его жёлтым.

Во втором файле style_btn_press.xml добавим точно такие же параметры, только изменим цвет заливки на оранжевый:
<*?***xml version="1.0" encoding="utf-8"***?*>
<**shape**
xmlns:android="http://schemas.android.com/apk/res/android">

<**stroke**
android:width="5dp"
android:color="@color/orange">
</**stroke**>

```
<corners
android:radius="20dp">
</corners>


<solid
android:color="@color/orange">
</solid>


</shape>
```

В третьем файле задаём стиль для нажатой и не нажатой кнопки, который мы только что определили в первых двух файлах:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
<item
android:state_pressed="false"
android:drawable="@drawable/style_btn_no_press">
</item>


<item
android:state_pressed="true"
android:drawable="@drawable/style_btn_press">
</item>


</selector>
```

Таким образом когда кнопка не нажата, будет применяться стиль из файла style_btn_no_press, а когда нажата – из файла style_btn_press.

Но и это ещё не всё. Чтобы данный стиль применить к конкретной кнопке, необходимо указать его в файле themes.xml в параметре

background темы btn_style1:

<**item name="android:background"**>@drawable/style_btn</**item**>

Теперь не нажатая кнопка у нас выглядит вот так:

А нажатая вот так:

Шпаргалка № 12. Анимированная кнопка.

Есть ещё один вариант, как красиво оформить кнопку и сделать её анимированной. Например, сделать её в виде фрукта, овоща, какого-либо предмета или даже картинки.

Смысл добавления анимации, следующий: при нажатии, кнопка будет становиться чуть больше или чуть меньше по размеру, таким образом пользователь будет уверен, что действительно нажал на неё, а после того, как он её отпустит – кнопка вернётся в прежнее состояние.

Для этого нужно подобрать подходящую и добавить в папку drawable. Назовём её button. Если это будет картинка с фоном, можете сделать её с расширением .jpg, а если без фона, то обязательно .png для прозрачного заднего фона.

Меняем строку в файле themes.xml в параметре background темы btn_style1:

<**item name="android:background"**>@drawable/button</**item**>

Сейчас кнопка будет выглядеть следующим образом:

Теперь добавим анимацию. Кликнем правой кнопкой мыши по папке res, далее New → Directory, папку называем anim. В этой папке создаём файл, нажимаем New → Animation Resource File, называем его button_animation.

Внутри данного файла добавляем следующий код, если хотим, чтобы кнопка при нажатии увеличивалась:

<?**xml version="1.0" encoding="utf-8"**?>

<**set xmlns:android="http://schemas.android.com/apk/res/android"**

android:interpolator="@android:anim/linear_interpolator">
<**scale**
android:fromXScale="1.0"
android:toXScale="1.5"
android:fromYScale="1.0"
android:toYScale="1.5"
android:pivotX="50%"
android:pivotY="50%"
android:duration="100"
android:repeatCount="1"
android:repeatMode="reverse"/>
</**set**>

Или такой, если хотим, чтобы она уменьшалась:
<*?***xml version="1.0" encoding="utf-8"***?*>
<**set xmlns:android="http://schemas.android.com/apk/res/android"**
android:interpolator="@android:anim/linear_interpolator">
<**scale**
android:fromXScale="1.0"
android:toXScale="0.5"
android:fromYScale="1.0"
android:toYScale="0.5"
android:pivotX="50%"
android:pivotY="50%"
android:duration="100"
android:repeatCount="1"
android:repeatMode="reverse"/>
</set>

Чтобы анимация начала работать переходим в MainActivity и добавляем две строки. Первой объявляем, что у нас есть некая анимация, которую мы хотим применить, прописываем её перед строкой, с помощью которой мы находим кнопку:

final Animation anim_button = AnimationUtils.*loadAnimation*(**this**, R.anim.***button_animation***);

А вторую перед намерением перейти на другую активность. Этой строкой мы говорим системе, что хотим нашу анимацию применить к данной кнопке до того, как она сработает и перебросит пользователя на другую активность.

v.startAnimation(anim_button);

Итого получаем:

```
public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

final Animation anim_button = AnimationUtils.loadAnimation(this, R.anim.button_animation);

Button start = (Button) findViewById(R.id.btn_start);

start.setOnClickListener(new View.OnClickListener() {

@Override
public void onClick(View v) {

v.startAnimation(anim_button);

try {
Intent intent = new Intent(MainActivity.this, Activity2.class);
startActivity(intent);finish();
```

```
} catch (Exception e){


}
}
});


}
}
```

Теперь наша кнопка не только красивая и имеет необычный вид, но и двигается при нажатии на неё пользователем.

Шпаргалка № 13. Системная кнопка «Назад».

Если мы делаем переход из одной активности в другую, то нужно обязательно сделать и переход из второй активности обратно в первую. Сделаем это с помощью системной кнопки «Назад» вот таким коротеньким кодом в java-файле Activity2:

```
@Override
public void onBackPressed(){


try {
Intent intent = new Intent(Activity2.this, MainActivity.class);
startActivity(intent);finish();
} catch (Exception e){
}
}
```

Должна получиться вот такая картина:

```java
package com.kta.mytestproject;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;

public class Activity2 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity2);
    }

    @Override
    public void onBackPressed(){

        try {
            Intent intent = new Intent( packageContext: Activity2.this, MainActivity.class);
            startActivity(intent);finish();
        } catch (Exception e){
        }
    }
}

```

В этом случае, если нажать на системную кнопку назад, то пользователь вернётся на приветственную активность (MainActivity).

Шпаргалка № 14. Выход из приложения двойным кликом.

Очень важно сделать выход из приложения двойным кликом по системной кнопке «Назад», с обязательным предупреждением об этом пользователя во всплывающем окошке.

Делать это нужно в главной активности MainActivity. Для начала объявим переменные для всплывающего окна до метода onCreate():

private long backPressedTime;

private Toast **backToast**;

А в самом конце дополним методом onBackPressed():

@Override

public void onBackPressed() {

if (**backPressedTime** + 2000 > System.*currentTimeMillis*()){

```
    backToast.cancel();
    super.onBackPressed();
    return;
    } else{
    backToast = Toast.makeText(getBaseContext(), "Нажмите ещё раз, чтобы выйти", Toast.LENGTH_SHORT);
    backToast.show();
    }
    backPressedTime = System.currentTimeMillis();
    }
```

Данный метод подразумевает следующее: пользователь на главной активности нажимает системную кнопку и тут же получает сообщение во всплывающем окошке: "Нажмите ещё раз, чтобы выйти", нажимает ещё раз и приложение закрывается. Если пользователь не нажмёт второй раз, выход не произойдёт.

Сделано это для того, чтобы приложение не закрывалось, если пользователь не хотел выходить, а нажал на кнопку случайно.

Шпаргалка № 15. Чек-бокс.

В этой главе рассмотрим такую нужную вещь, как чек-бокс. Это когда из доступных вариантов можно выбрать один или несколько, поставить галочку там, где это необходимо. Рассмотрим на практическом примере.

Добавим на экран несколько текстовых строк и чек-боксов для выбора. Например, нам нужно сделать заказ пиццы и выбрать понравившиеся ингредиенты. Создадим строковые ресурсы.

```
<string name="ing1">Соус</string>
<string name="ing2">Томаты</string>
<string name="ing3">Лук</string>
<string name="ing4">Ветчина</string>
<string name="ing5">Куриная грудка</string>
<string name="ing6">Маслины</string>
<string name="ing7">Сыр</string>
<string name="ing8">Укроп</string>
```

Для текстовых полей создадим стиль такой:

```
<style name="text_style2">
<item name="android:layout_width">wrap_content</item>
<item name="android:layout_height">wrap_content</item>
<item name="android:textColor">@color/black</item>
<item name="android:textSize">18sp</item>
<item name="android:textAllCaps">false</item>
<item name="android:textStyle">bold|italic</item>
<item name="android:gravity">left</item>
<item name="android:padding">10dp</item>
<item name="android:typeface">serif</item>
</style>
```

А для чек-боксов такой:

```
<style name="checkbox_style">
<item name="android:layout_width">wrap_content</item>
<item name="android:layout_height">wrap_content</item>
<item name="android:textSize">18sp</item>
</style>
```

Далее создадим текстовые поля в макете LinearLayout. Макетов будет несколько: один основной вертикальный, а в него вложим 8 горизонтальных, чтобы уместить чек-боксы и тексты, им соответствующие.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
```

android:layout_height="wrap_content"
android:orientation="vertical">

<**LinearLayout**
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal">
<**CheckBox**
android:id="@+id/ch_box1"
style="@style/checkbox_style"/>
<**TextView**
android:id="@+id/tv1"
android:text="@string/ing1"
style="@style/text_style2"/>
</**LinearLayout**>

</**LinearLayout**>

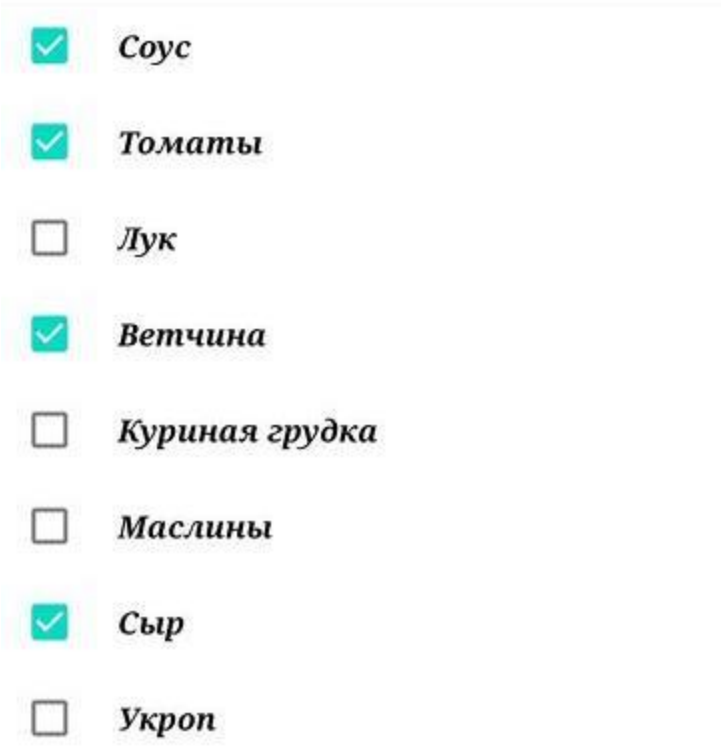
Далее копируем горизонтальный LinearLayout и вместе с его содержимым вставляем ещё семь раз, меняя только названия на единичку больше, например:

ch_box2, ch_box3 … ch_box8
tv2, tv3 … tv8
ing2, ing3 … ing8

Теперь у нас есть некий список, и флажки, которые можно отметить.

- ☐ *Соус*
- ☐ *Томаты*
- ☐ *Лук*
- ☐ *Ветчина*
- ☐ *Куриная грудка*
- ☐ *Маслины*
- ☐ *Сыр*
- ☐ *Укроп*

В выбранном состоянии чек-боксы будут выглядеть так:



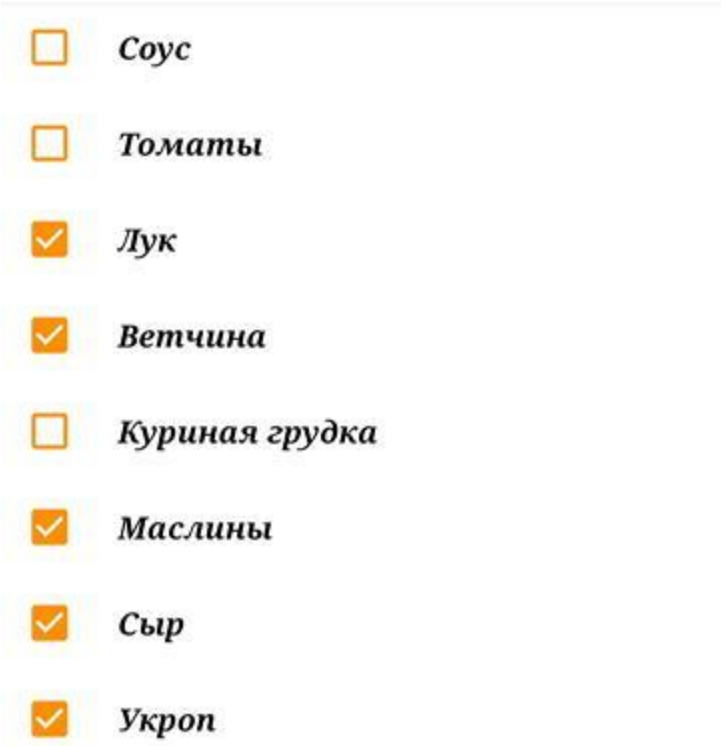
Шпаргалка № 16. Чек-бокс другого цвета.

По умолчанию чек-бокс в Android Studio серого цвета, а при выборе – бирюзовый. Но что, если Вы хотите сделать его синим? А может быть жёлтым или оранжевым?

Это не проблема. Достаточно добавить одну строку в стиль:

<**item name="android:buttonTint"**>@color/orange</**item**>

Тогда Ваши чек-боксы будут выглядеть совсем по-другому.



Шпаргалка № 17. Чек-бокс своего стиля.

При желании можно создать свой стиль, например из картинки. Для этого создайте две свои картинки, размером примерно 50*50 пикселей, формата .png и вставьте в папку drawable. Там же создайте xml-файл с

названием checkbox_selector, а в нём добавьте код, который будет определять какая картинка будет отображаться на невыбранном чек-боксе, а какая на выбранном.

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/false_press"
android:state_checked="false"/>
<item android:drawable="@drawable/true_press"
android:state_checked="true"/>

</selector>
```

Далее в стиле для чек-бокса меняем строку:
<**item name="android:buttonTint"**>@color/orange</**item**>
На строку:
<**item name="android:button"**>@drawable/checkbox_selector</**item**>

А в макете, где прописаны наши чек-боксы

```
<CheckBox
android:id="@+id/ch_box1"
style="@style/checkbox_style"/>
```

Меняем на:

```
<androidx.appcompat.widget.AppCompatCheckBox
android:id="@+id/ch_box1"
style="@style/checkbox_style"/>
```

Таким образом Вы создадите свой собственный авторский стиль для чек-боксов, чтобы Ваше приложение приобрело индивидуальность и красивый вид.

Шпаргалка № 18. Собственный шрифт.

Мы уже пробовали менять шрифт, который нам предлагает Android Studio, но что же делать, если хочется применить свой собственный, красивый?

Всё очень просто. Скачиваем из интернета любой понравившийся шрифт и сохраняем в папку под названием font.

В Android Studio нажимаем File → New → Folder → Assets Folder и создаём папку assets, в которую вставляем наш файл font.otf, который и будет отвечать за новый шрифт.

Создаём строковый ресурс, условно это будет:

<**string name="str4"**>Это очень красивый шрифт, я сам его создал!!!</**string**>

В xml-файле добавляем текстовое поле и называем его, например, tv_font.

В качестве ссылки на текст указываем:

android:text="@string/str4"

В новом java-файле Activity3 до метода onCreate() создаём переменную:

TextView **title**;

Создаём метод:

public void setFonts(){

Typeface type = Typeface.*createFromAsset*(getAssets(), **"font.otf"**);

title.setTypeface(type);

}

Импортируем класс Typeface и указываем, что хотим в качестве шрифта установить именно font.otf, а второй строкой указываем, что хотим применить его к title.

Внутри метода onCreate() ищем наше текстовое поле по ID:

title = (TextView) findViewById(R.id.***tv_font***);

И следующей строкой вызываем метод:

setFonts();

Должно получиться следующим образом:

```
public class Activity3 extends AppCompatActivity {

    TextView title;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity3);

        title = (TextView) findViewById(R.id.tv_font);
        setFonts();

    }


    public void setFonts(){
        Typeface type = Typeface.createFromAsset(getAssets(), path: "font.otf");

        title.setTypeface(type);
    }

}
```

Вот и готово! А на экране телефона пользователь увидит следующее:

Таким методом собственный шрифт можно применять не только к кнопкам и коротким надписям, но и к большим текстам, если у Вас в приложении такие имеются.

Шпаргалка № 19. Случайный выбор.

В Android Studio существует такое понятие как случайный выбор, или Random. Создать его совсем не сложно. Это могут быть цифры, слова или фразы и даже картинки. А также их можно объединять между собой. Например, мы делаем детское приложение и хотим показать, как выглядит животное, как оно говорит и как называется. При нажатии на кнопку, которую мы специально создадим, будет срабатывать случайный выбор и выдавать результат на экран. Рассмотрим подробнее.

Для начала создадим строковые ресурсы для названия животных:

<**string name="cat"**>Кошка</**string**>
<**string name="dog"**>Собака</**string**>
<**string name="cow"**>Корова</**string**>
<**string name="frog"**>Лягушка</**string**>
<**string name="duck"**>Утка</**string**>
Им должны соответствовать строки кто как говорит:
<**string name="g1"**>Мяу</**string**>
<**string name="g2"**>Гав</**string**>
<**string name="g3"**>Мууу</**string**>
<**string name="g4"**>Ква ква</**string**>
<**string name="g5"**>Кря кря</**string**>

И вставляем в папку drawable картинки, соответствующие названиям животных с именами:
cat.jpg
dog.jpg
cow.jpg
frog.jpg
duck.jpg

Чтобы использовать рандом, сначала необходимо все данные записать в массив. Для этого создаём java-класс под названием Array. Массива будет три:
Для названия животного
final int[] **arr1** = {
R.string.***cat***,
R.string.***dog***,
R.string.***cow***,
R.string.***frog***,
R.string.***duck***,
};

Для картинки
final int[] **arr2** = {

R.drawable.***cat***,
R.drawable.***dog***,
R.drawable.***cow***,
R.drawable.***frog***,
R.drawable.***duck***,
};

Как говорит животное
final int[] **arr3** = {
R.string.***g1***,
R.string.***g2***,
R.string.***g3***,
R.string.***g4***,
R.string.***g5***,
};

Для тестирования создадим новый класс Activity4 и разметку к нему activity4.xml, где создадим два текстовых поля, поле под картинку и кнопку «Далее» для последующего применения рандома.

Перед этим создайте свои стили и цвета для текста и кнопки. Например, задний фон можно сделать зеленым, а кнопку жёлтой.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/green3"
android:orientation="vertical">


<TextView
android:id="@+id/name"
style="@style/text_style3"/>
```

<**ImageView**
android:id="@+id/picture"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:scaleType="centerCrop"/>

<**TextView**
android:id="@+id/voic"
style="@style/text_style3"/>

<**androidx.appcompat.widget.AppCompatButton**
android:id="@+id/btn_start"
android:text="@string/str5"
style="@style/btn_style2"/>

</**LinearLayout**>

В java-файле создаём переменную, для наших животных, например:
public int animal;

Создаём новый объект из класса Array:
Array **array** = **new** Array();

И рандом для генерации случайных чисел:
Random **random** = **new** Random();

Создадим метод rand_anim():
public void rand_anim (){
final TextView name = (TextView)findViewById(R.id.***name***);

```
final ImageView picture = (ImageView)findViewById(R.id.picture);
final TextView voic = (TextView)findViewById(R.id.voic);
animal = random.nextInt(5);
name.setText(array.arr1[animal]);
picture.setImageResource(array.arr2[animal]);
voic.setText(array.arr3[animal]);
}
```

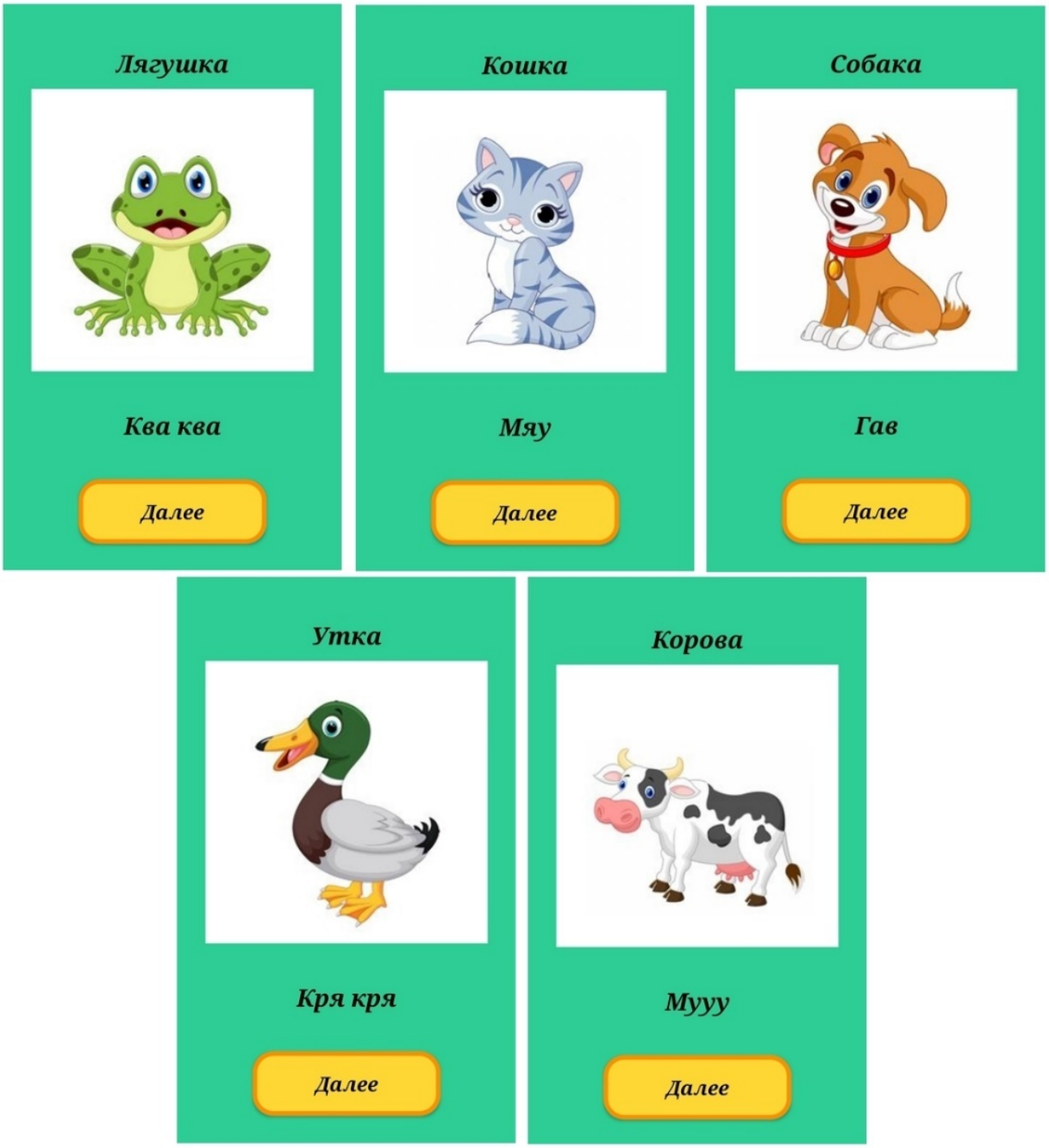
В этом методе мы первыми тремя строками находим наши текстовые поля и картинку, далее создаём рандом указав количество равное количеству наших строк, в данном случае это 5, а в последующих трёх строках говорим системе, что хотим присвоить случайный выбор именно этим полям, взяв строку из массива arr1, arr2 и arr3, которые созданы в классе Array.

Заполняем метод onCreate(). Как Вы помните, данный метод выполняется сразу же при загрузке, поэтому, чтобы наш рандом сработал автоматически, выполним его первой же строкой.

```
rand_anim();
```

А затем заставим кнопку, по нажатию на неё, выполнять это же самое действие:

```
Button next = (Button) findViewById(R.id.btn_next);
next.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
rand_anim();
}
});
```



Таким образом, при нажатии на кнопку «Далее» срабатывает наш случайный выбор, и в поля подставляются различные животные, в соответствии с их названиями и голосами. А код должен выглядеть вот так:

```
public class Activity4 extends AppCompatActivity {

    public int animal; //переменная для орла и решки
    Array array = new Array(); //создали новый объект из класса Array
    Random random = new Random(); //для генерации случайных чисел

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity4);

        rand_anim();

        Button next = (Button) findViewById(R.id.btn_next);
        next.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                rand_anim();
            }
        });
    }

    public void rand_anim (){
        final TextView name = (TextView)findViewById(R.id.name);
        final ImageView picture = (ImageView)findViewById(R.id.picture);
        final TextView voic = (TextView)findViewById(R.id.voic);
        animal = random.nextInt( bound: 5);
        name.setText(array.arr1[animal]);
        picture.setImageResource(array.arr2[animal]);
        voic.setText(array.arr3[animal]);
    }

}
```

Шпаргалка № 20. Изменение цвета текста одним касанием.

Мы уже рассмотрели метод setOnClickListener(), который срабатывает при нажатии на кнопку и способен выполнять различные

функции. В этом разделе мы рассмотрим немного другой метод, setOnTouchListener(), который способен выполнять две различные задачи, первую – при нажатии на кнопку, а вторую – в тот момент, когда пользователь уберёт палец.

Рассмотрим на примере: допустим, мы создали приложение с загадками, на экране мы видим загадку, а ответ от глаз скрыт. Посмотреть его можно только если нажать на кнопку. Пока кнопка нажата – ответ виден. Как только пользователь отпускает кнопку – ответ скрывается.

Для тестирования создадим новый java-класс Activity5 и макет activity.xml, в который поместим два текстовых поля и одну кнопку.

Создадим текстовые ресурсы, загадку, ответ на неё и текст для кнопки:

<**string name="str6"**>Не лает, не кусает, а в дом не пускает? </**string**>

<**string name="str7"**>Замок</**string**>

<**string name="str8"**>Показать ответ</**string**>

Для того, чтобы скрыть текст, сделаем его прозрачным:

<**color name="transparent"**>#00FFFFFF</**color**>

В макете прописываем:

<?**xml version="1.0" encoding="utf-8"**?>

<**LinearLayout**
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/green3"
android:orientation="vertical">

<**TextView**
android:id="@+id/zagadka"
android:text="@string/str6"

style="@style/text_style3"/>

<**TextView**
android:id="@+id/otvet"
android:text="@string/str7"
android:textColor="@color/transparent"
style="@style/text_style3"/>

<**androidx.appcompat.widget.AppCompatButton**
android:id="@+id/btn_otv"
android:text="@string/str8"
style="@style/btn_style2"/>

</**LinearLayout**>

А в java-файле прописываем код в методе onCreate():

TextView otvet = (TextView) findViewById(R.id.***otvet***);

Button answer = (Button) findViewById(R.id.***btn_otv***);

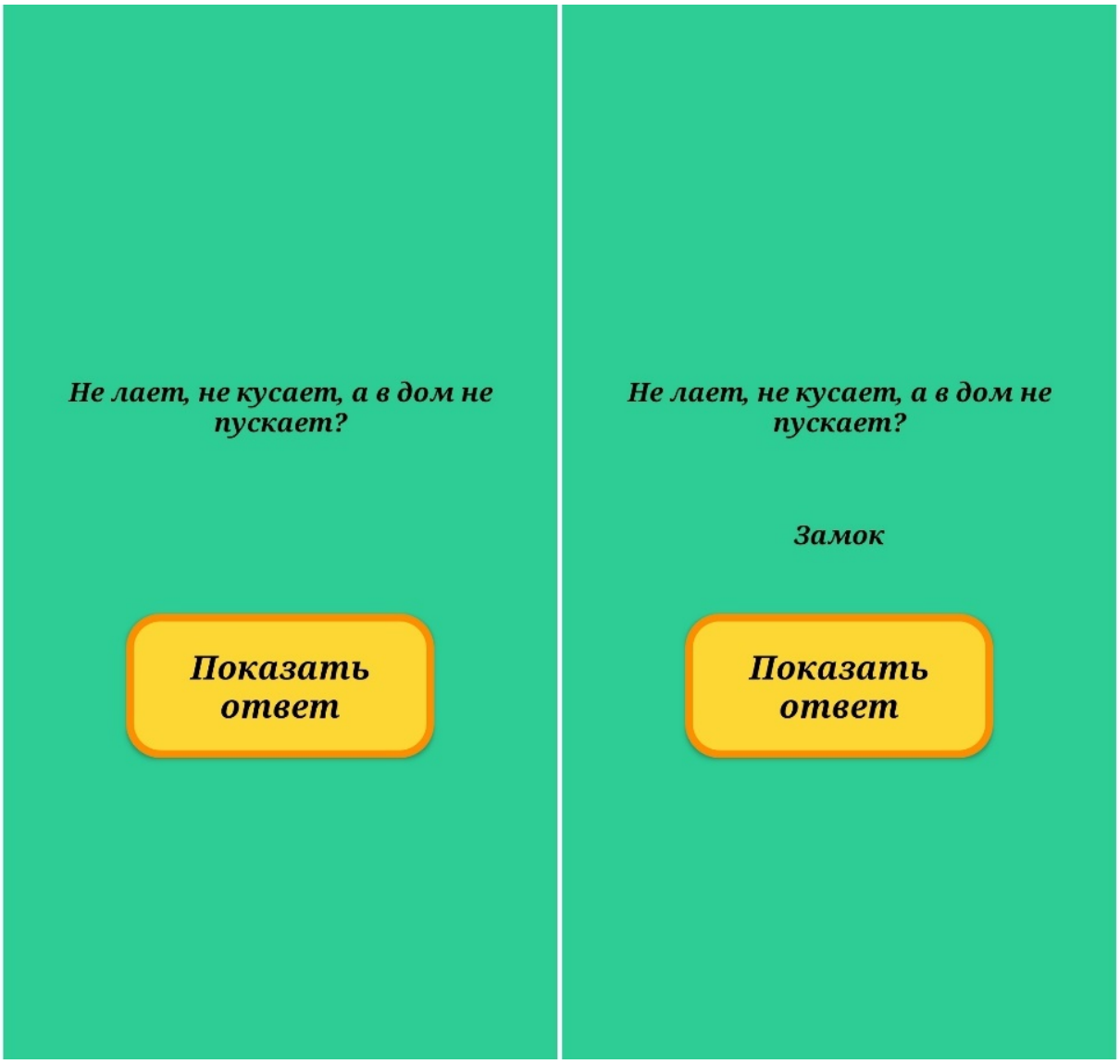
answer.setOnTouchListener(**new** View.OnTouchListener() {

@Override
public boolean onTouch(View v, MotionEvent event) {

if (event.getAction() == MotionEvent.***ACTION_DOWN***) {

```
otvet.setTextColor(getResources().getColor(R.color.black));


} else if (event.getAction() == MotionEvent.ACTION_UP) {
otvet.setTextColor(getResources().getColor(R.color.transparent));
}


return true;


}
});
```

Таким образом мы говорим Android Studio, что если пользователь прикоснулся к кнопке, то текст ответа нужно сделать чёрным, а если отпустил палец – то снова прозрачным.



По подобной схеме можно сделать всё, что угодно: поменять цвет текста, цвет заднего фона, цвет кнопки, можно изменить непосредственно сам текст или какую-либо картинку, в общем, всё, что Вы захотите.

```
public class Activity5 extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity5);

        TextView otvet = (TextView) findViewById(R.id.otvet);

        Button answer = (Button) findViewById(R.id.btn_otv);

        answer.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {

                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    otvet.setTextColor(getResources().getColor(R.color.black));
                } else if (event.getAction() == MotionEvent.ACTION_UP) {
                    otvet.setTextColor(getResources().getColor(R.color.transparent));
                }

                return true;
            }
        });
    }
}
```

Основные методы:

setText(R.string.name) – присвоить другой текст;
setTextColor(getResources().getColor(R.color.name)) – изменить цвет текста;
setBackgroundResource(R.drawable.name) – изменить задний фон;
setImageResource(R.drawable.name) – изменить картинку;
startAnimation(name) – запустить анимацию.

Список далеко не полный, всё зависит от того, что именно Вы хотите изменить в своём приложении.

Шпаргалка № 21. Диалоговое окно.

Во многих приложениях есть обратная связь для пользователей в виде диалогового окна. Чаще всего в нём выводится какой-либо вопрос или информация. Например, о том, что пользователь набрал определённое количество баллов или система уточняет, действительно ли пользователь желает удалить данные. На этом примере и рассмотрим.

Представим, что в приложении есть некий блок EditText, в который пользователь вводит данные, но потом хочет удалить их нажатием одной кнопки.

Создадим новый java-файл Activity6 и разметку activity6.xml, в которую добавим текстовое поле EditText и кнопку.

Добавим строковые ресурсы для текстового поля, кнопки и диалогового окна:

<**string name="str9"**>Заполните данные</**string**>

<**string name="str10"**>Удалить всё</**string**>

<**string name="str11"**>Вы действительно хотите удалить все данные?</**string**>

<**string name="str12"**>Да</**string**>

<**string name="str13"**>Нет</**string**>

Создадим диалоговое окно: нажмём правой кнопкой мыши по папке layout, далее New → Layout Resource File, назовём его dialog.xml и добавим следующий код:

<?**xml version="1.0" encoding="utf-8"**?>

<**LinearLayout**
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/dialogfon"
android:background="@color/orange"
android:layout_gravity="center"
android:scaleType="centerCrop"

android:layout_margin="20dp">

<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="20dp"
android:gravity="center">

<TextView
android:id="@+id/text_dialog"
android:text="@string/str11"
style="@style/text_style3"/>

<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:padding="20dp"
android:gravity="center">

<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_yes"
android:text="@string/str12"
style="@style/btn_style3"/>

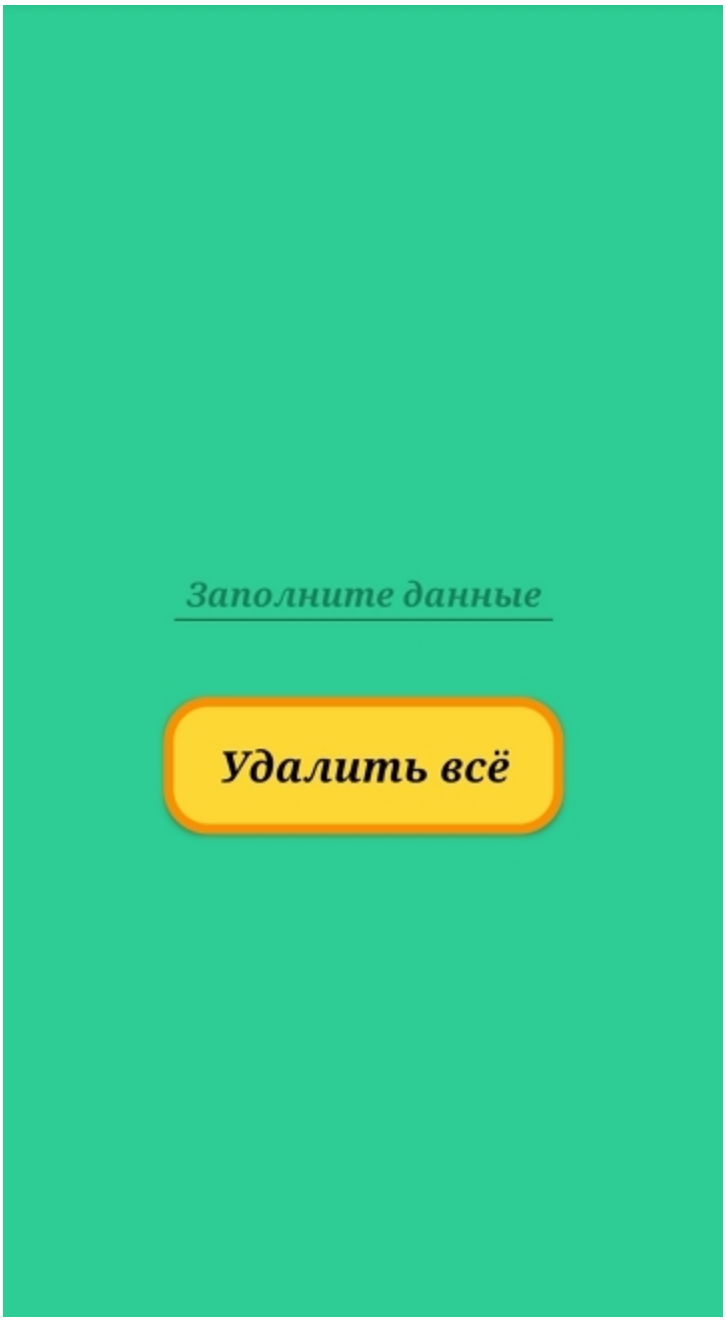
<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_no"
android:text="@string/str13"
style="@style/btn_style3"/>

```
</LinearLayout>
</LinearLayout>
</LinearLayout>
```

В разметке activity6.xml добавим текстовое поле и кнопку для удаления данных:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/green3"
android:orientation="vertical">


<EditText
android:id="@+id/text"
android:hint="@string/str9"
style="@style/text_style3"/>


<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_del"
android:text="@string/str10"
style="@style/btn_style2"/>


</LinearLayout>
```

Получим такой результат:



У нас есть поле для ввода и кнопка для удаления данных. Сейчас пропишем код, который будет вызывать диалоговое окно. В нём будет вопрос, действительно ли пользователь хочет всё удалить и два варианта ответа: да или нет. При нажатии на кнопку «Да», данные будут удалены и диалоговое окно закрыто. При нажатии на кнопку «Нет» данные будут сохранены, а диалоговое окно просто закроется.

Чтобы данный метод работал, в java-файле прописываем следующий код. Для начала создадим переменную для диалогового окна:

Dialog **dialog**;

Затем в методе onCreate() описываем логику событий:

dialog = **new** Dialog(**this**

dialog.setContentView(R.layout.***dialog***);

dialog.getWindow().setBackgroundDrawable(**new** ColorDrawable(Color.***TRANSPARENT***));

dialog.setCancelable(**false**);

Первой строкой создаём диалоговое окно;

Второй строкой указываем путь к макету диалогового окна;

Третьей устанавливаем прозрачный фон, так как у нас уже есть свой дизайн;

Четвёртой указываем, что диалоговое окно нельзя закрыть системной кнопкой «Назад».

Далее находим все наши элементы:

TextView text = (TextView) findViewById(R.id.***text***);

Button delete = (Button) findViewById(R.id.***btn_del***);

Button yes = (Button) **dialog**.findViewById(R.id.***btn_yes***);

Button no = (Button) **dialog**.findViewById(R.id.***btn_no***);

И теперь по очереди задаём действия, которые будут совершаться по нажатию на кнопки. Как всегда, используем метод setOnClickListener().

```
delete.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
dialog.show();
}
});
```

```
yes.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
text.setText("");
dialog.dismiss();
```

```
}
});


no.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
dialog.dismiss();
}
});
```

dialog.show(); – показываем диалоговое окно;
text.setText(""); – стираем данные из текстового поля;
dialog.dismiss(); – закрываем диалоговое окно.

Вот такое замечательное диалоговое окно у нас получилось. При желании и необходимости его можно дополнить картинками, красивым задним фоном и ещё чем пожелаете. Это такой же layout и оформляется как любой другой макет.

Шпаргалка № 22. Проверка на равенство.

В этой главе мы рассмотрим такую замечательную возможность, как проверку на равенство или проверку на правильность, смотря для чего Вы используете данный метод. Называется он equals() и выполняет проверку содержимого текстового поля.

Рассмотрим на примере. Допустим, Вы делаете приложение на проверку знаний по математике и задаёте пример, сколько будет x+y? У пользователя есть поле, куда он должен ввести ответ, и кнопка для проверки правильности своего ответа.

Создадим новый макет и строковые ресурсы:

<**string name="str14"**>Сколько будет 25+20?</**string**>
<**string name="str15"**>Введите ответ</**string**>
<**string name="str16"**>Проверить</**string**>

Layout заполним двумя текстовыми полями и кнопкой:

<?**xml version="1.0" encoding="utf-8"**?>
<**LinearLayout**
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/green3"
android:orientation="vertical">

<**TextView**
android:id="@+id/primer"
android:text="@string/str14"
style="@style/text_style3"/>

<**EditText**
android:id="@+id/otvet"
android:hint="@string/str15"
style="@style/text_style3"/>

<**androidx.appcompat.widget.AppCompatButton**
android:id="@+id/btn_proverka"

android:text="@string/str16"
style="@style/btn_style2"/>

</**LinearLayout**>

В java-файле создаём переменные для всплывающего сообщения в случае правильного ответа и в случае неправильного ответа, чтобы сообщить пользователю, ошибся он или нет.

private Toast **trueToast**;
private Toast **falseToast**;

Далее определяем наше текстовое поле и кнопку для проверки на равенство:

EditText otvet = (EditText) findViewById(R.id.***otvet***);
Button proverka = (Button) findViewById(R.id.***btn_proverka***);

И задаём кнопке определённое действие с помощью конструкции if-else, то есть проверяем, если содержимое EditText равно заданному значению, всплывает сообщение «Всё верно», а если нет – «Неверно, попробуйте ещё раз».

proverka.setOnClickListener(**new** View.OnClickListener() {
@Override
public void onClick(View v) {

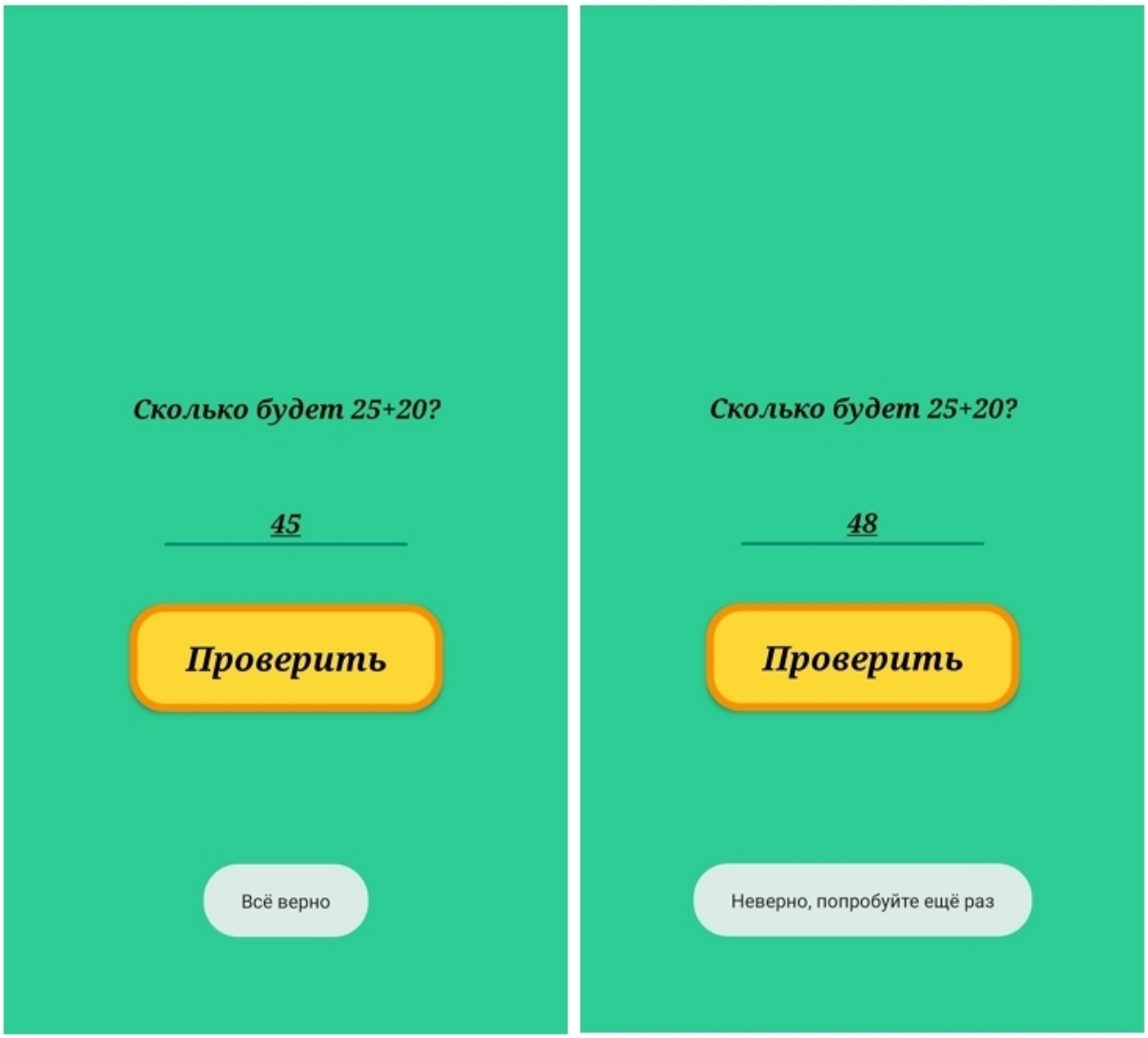
if (otvet.getText().toString().equals(**"45"**)) {
trueToast = Toast.*makeText*(getBaseContext(), **"Всё верно"**, Toast.***LENGTH_SHORT***);
trueToast.show();
} **else** {
falseToast = Toast.*makeText*(getBaseContext(), **"Неверно**, **попробуйте ещё раз"**, Toast.***LENGTH_SHORT***);
falseToast.show();

```
    }


    }
    });
```

Полный код:

```
public class Activity7 extends AppCompatActivity {

    private Toast trueToast;
    private Toast falseToast;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity7);

        EditText otvet = (EditText) findViewById(R.id.otvet);
        Button proverka = (Button) findViewById(R.id.btn_proverka);

        proverka.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if (otvet.getText().toString().equals("45")) {
                    trueToast = Toast.makeText(getBaseContext(), text: "Всё верно", Toast.LENGTH_SHORT);
                    trueToast.show();
                } else {
                    falseToast = Toast.makeText(getBaseContext(), text: "Неверно, попробуйте ещё раз", Toast.LENGTH_SHORT);
                    falseToast.show();
                }

                }
        });

    }
}
```

Проверяем данный метод на устройстве и убеждаемся, что всё отлично работает:



Таким образом на равенство можно проверять не только числа, но и текст.

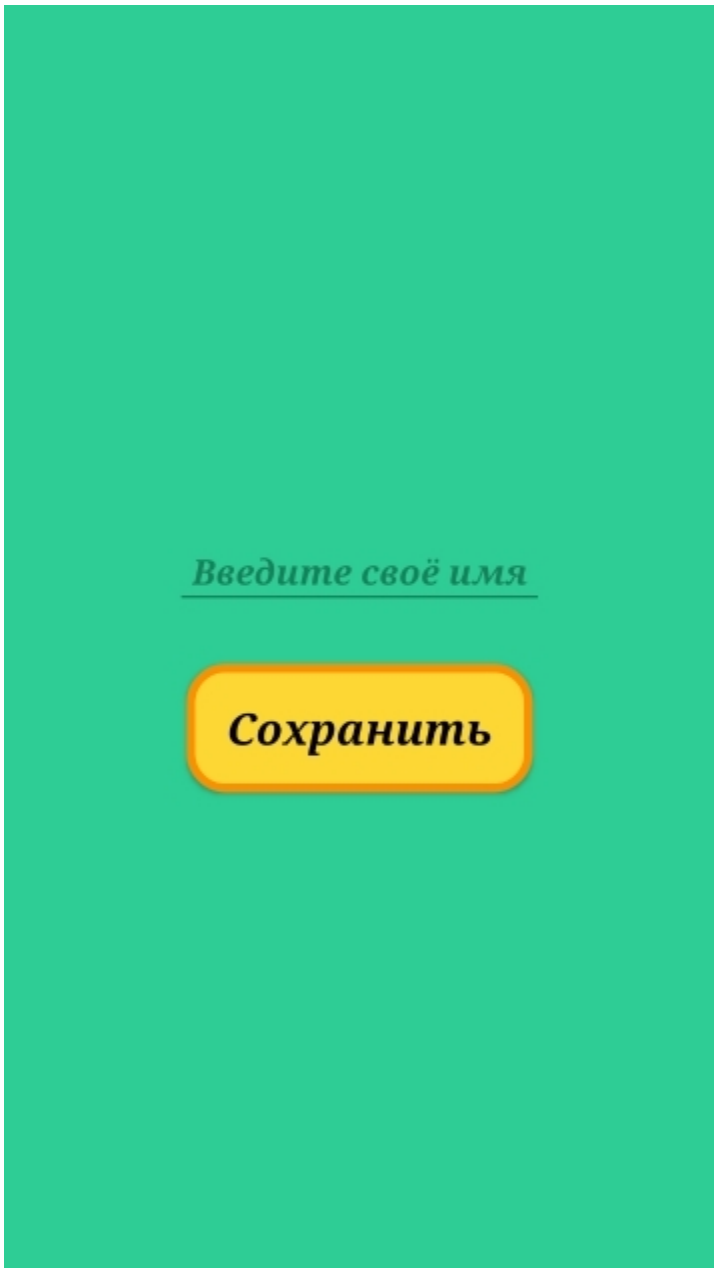
Шпаргалка № 23. Сохранение данных.

При использовании EditText нужно предполагать, что пользователь вводит какие-то данные и хочет их сохранить. Это мы и сделаем с помощью SharedPreferences.

Например, пользователю нужно ввести своё имя и сохранить его (на самом деле, это может быть что угодно, вплоть до больших текстов, цифр и прочего). Создаём макет:

<?**xml version="1.0" encoding="utf-8"**?>

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/green3"
android:orientation="vertical">


<EditText
android:id="@+id/name"
android:hint="@string/str2"
style="@style/text_style3"/>


<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_save"
android:text="@string/str17"
style="@style/btn_style2"/>


</LinearLayout>
```



В java-файле добавляем переменные:
private EditText **name**;
SharedPreferences **sPref**;
final String **SAVED_TEXT** = **"saved_text"**;

Создаём два метода, один для сохранения данных:
void saveText() {

```
sPref = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor ed = sPref.edit();
ed.putString(SAVED_TEXT, name.getText().toString());
ed.apply();
}
```

И второй для загрузки сохранённый данных:

```
void loadText() {
sPref = getPreferences(MODE_PRIVATE);
String savedText = sPref.getString(SAVED_TEXT, "");
name.setText(savedText);
}
```

В методе onCreate() прописываем:

```
name = (EditText) findViewById(R.id.name);
loadText();
```

```
Button save = (Button) findViewById(R.id.btn_save);
```

```
save.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
saveText();
}
});
```

В данном случае мы говорим системе, что по нажатию на кнопку «Сохранить» выполняется метод сохранения данных, а при загрузке страницы при следующем запуске приложения – метод загрузки.

```java
private EditText name;
SharedPreferences sPref;
final String SAVED_TEXT = "saved_text";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity8);

    name = (EditText) findViewById(R.id.name);
    loadText();

    Button save = (Button) findViewById(R.id.btn_save);
    save.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
           saveText();
            }
    });
}

void saveText() {
    sPref = getPreferences(MODE_PRIVATE);
    SharedPreferences.Editor ed = sPref.edit();
    ed.putString(SAVED_TEXT, name.getText().toString());
    ed.apply();
}

void loadText() {
    sPref = getPreferences(MODE_PRIVATE);
    String savedText = sPref.getString(SAVED_TEXT, s1: "");
    name.setText(savedText);
}
```

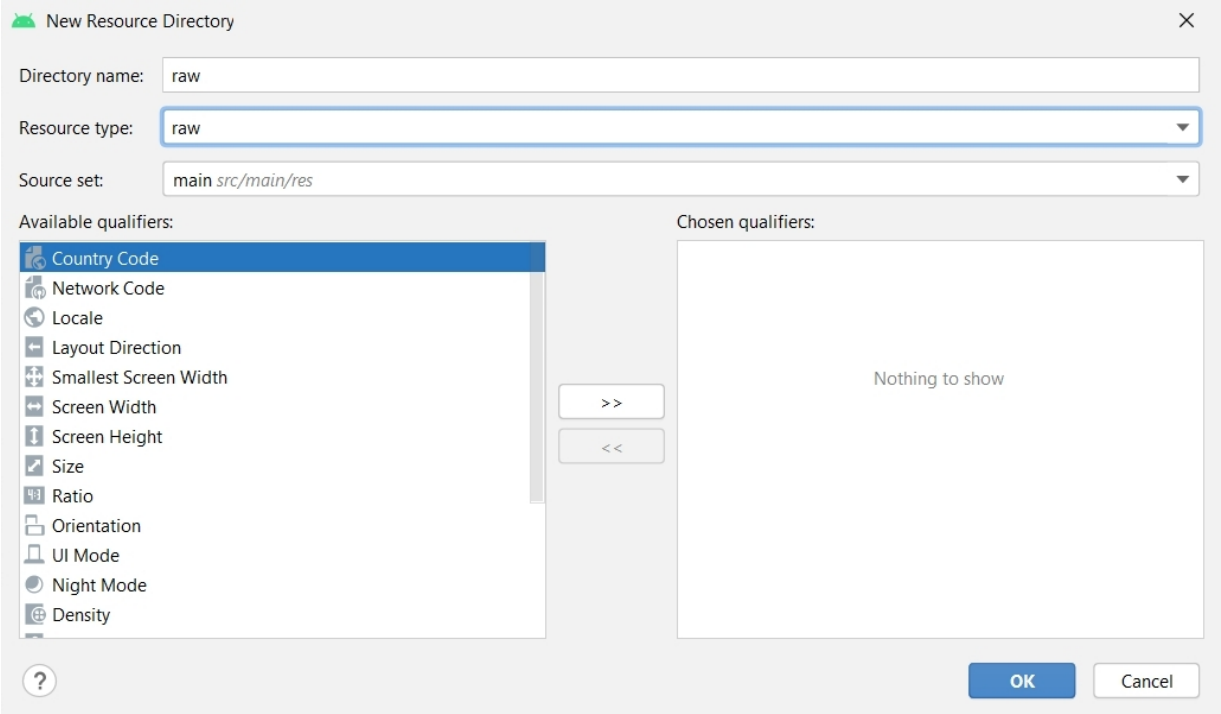
Шпаргалка № 24. Добавление звука и музыки.

Многие приложения содержат в себе различные звуки и музыку, и именно в этом разделе мы об этом поговорим. О том, как их правильно добавить и как с помощью кнопки воспроизводить.

Добавим в проект две картинки, изображающие старт и стоп в папку drawable под названиями play и stop.

В папке res создадим новую папку, кликаем правой кнопкой мыши New → Android Resource Directory, выбираем raw.

Вставим в папку любую понравившуюся музыку (или звук) в формате mp3.



Создадим макет.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

```
android:gravity="center"
android:background="@color/orange"
android:orientation="vertical">


<TextView
android:id="@+id/music"
style="@style/text_style3"
android:text="@string/str18"/>


<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_play"
android:layout_width="50dp"
android:layout_height="50dp"
android:layout_margin="10dp"
android:background="@drawable/play"/>


<androidx.appcompat.widget.AppCompatButton
android:id="@+id/btn_stop"
android:layout_width="50dp"
android:layout_height="50dp"
android:layout_margin="10dp"
android:background="@drawable/stop"/>


</LinearLayout>
```

Теперь наша задача сделать так, чтобы кнопка btn_play воспроизводила нашу музыку (или звук), а кнопка btn_stop – останавливала.

Создадим переменную для музыки:

private MediaPlayer **my_music**;

Определяем кнопки для старта и остановки:

final Button play = (Button)findViewById(R.id.***btn_play***);

final Button stop = (Button)findViewById(R.id.***btn_stop***);

Указываем Android Studio на расположение музыки:

```
    my_music = MediaPlayer.create(this, R.raw.music);
    my_music.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
    my_music.start();
    }
    });
```
Прописываем действия для кнопок:
```
    play.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    my_music.start();
    }
    });


    stop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
    my_music.stop();
    }
    });
```

```
    private MediaPlayer my_music;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity9);

        final Button play = (Button)findViewById(R.id.btn_play);
        final Button stop = (Button)findViewById(R.id.btn_stop);

        play.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                my_music.start();
                }
        });

        stop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                my_music.stop();
            }
        });

        my_music = MediaPlayer.create( context: this, R.raw.music);
        my_music.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                my_music.start();
            }
        });
    }
}
```

Теперь наше приложение воспроизводит музыку и звук.

Шпаргалка № 25. Изменение цвета панели.

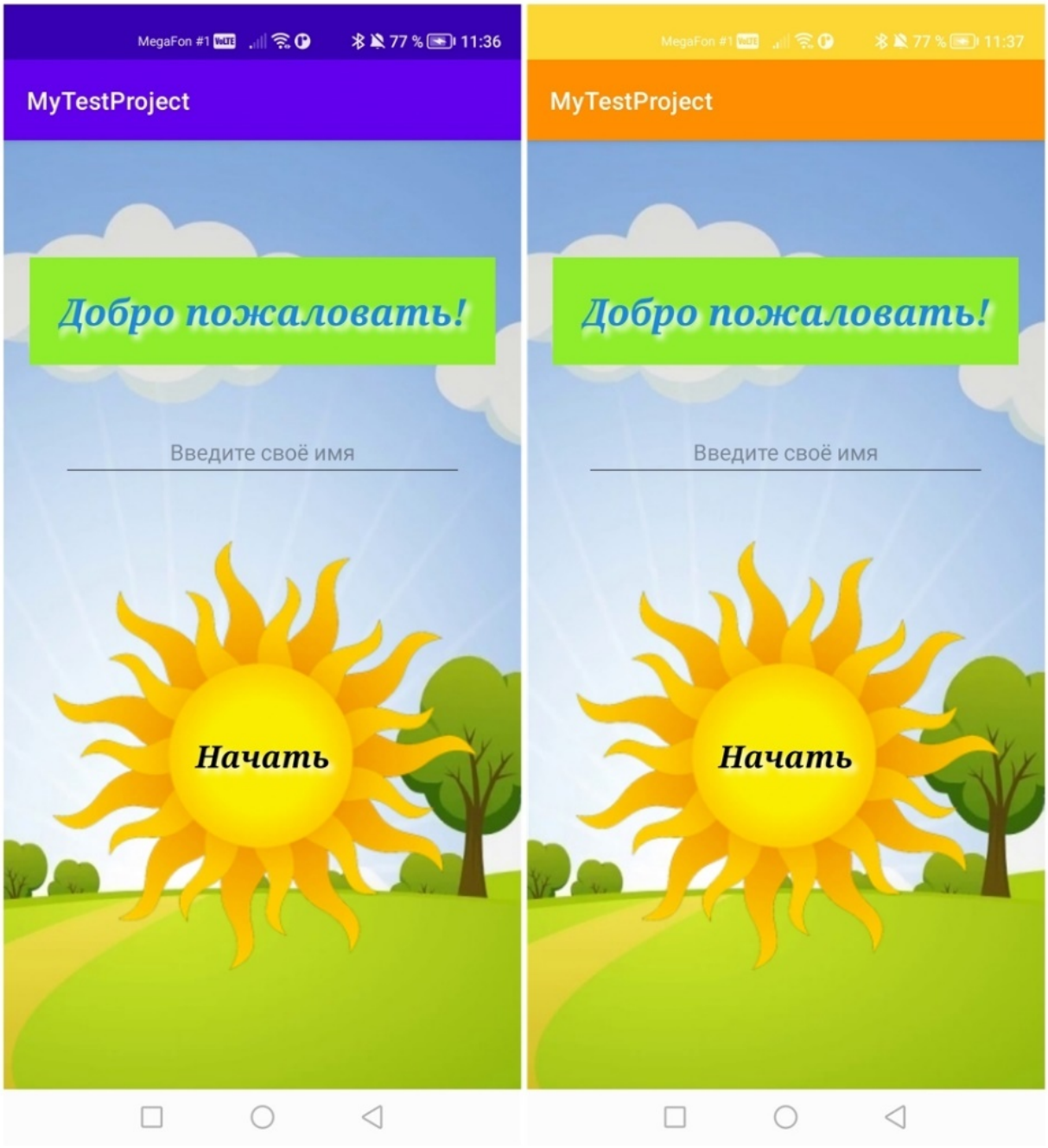
Чтобы изменить цвет панели достаточно лишь зайти в файл themes.xml и изменить цвет в строках:

<**item name="colorPrimary"**>@color/purple_500</**item**>
<**item name="colorPrimaryVariant"**>@color/purple_700</**item**>

По умолчанию они синие, а Вы можете изменить на любой понравившийся, например:

<**item name="colorPrimary"**>@color/orange</**item**>
<**item name="colorPrimaryVariant"**>@color/yellow</**item**>

Таким образом мы изменим и панель, и строку состояния телефона:



А если Вам совсем не нужна панель, то меняем строку

<style name="Theme.MyTestProject" parent="Theme.MaterialComponents.DayNight.DarkActionBar">

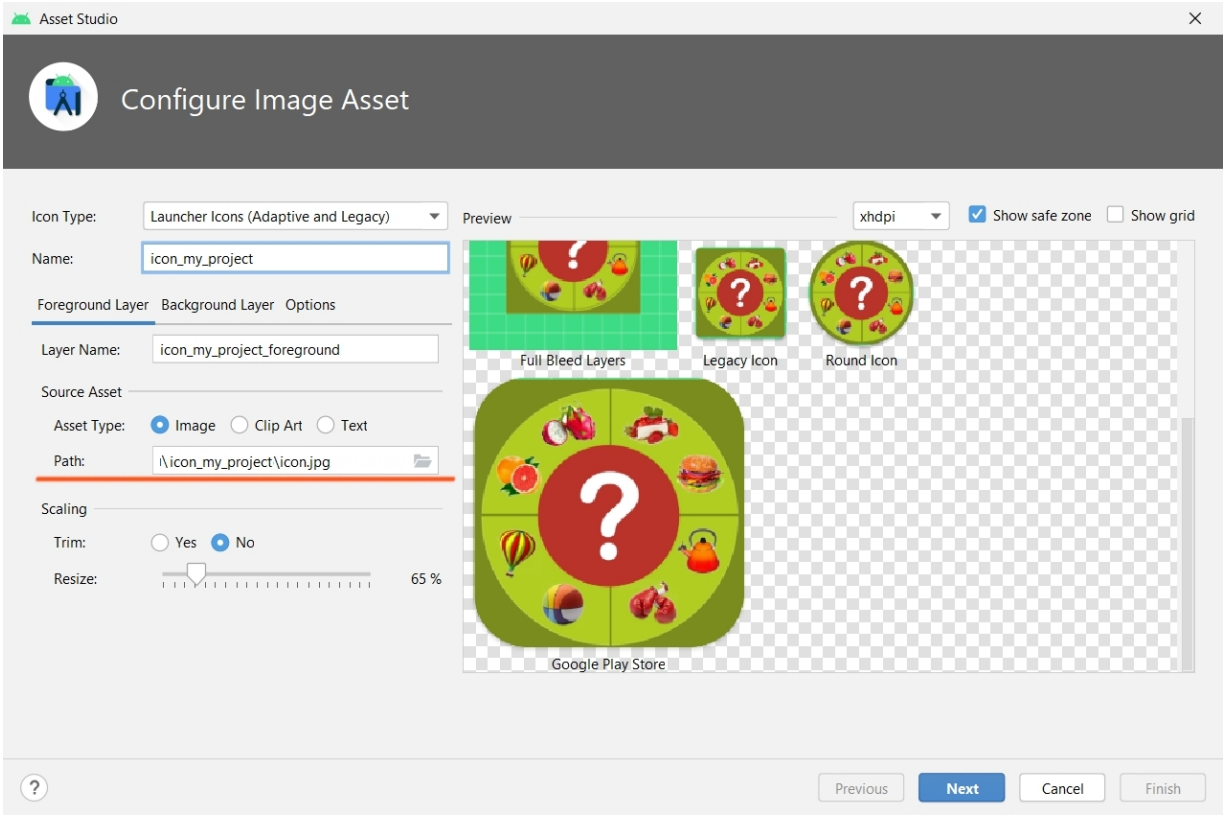
На строку

<style name="Theme.MyTestProject" parent="Theme.MaterialComponents.DayNight.NoActionBar">

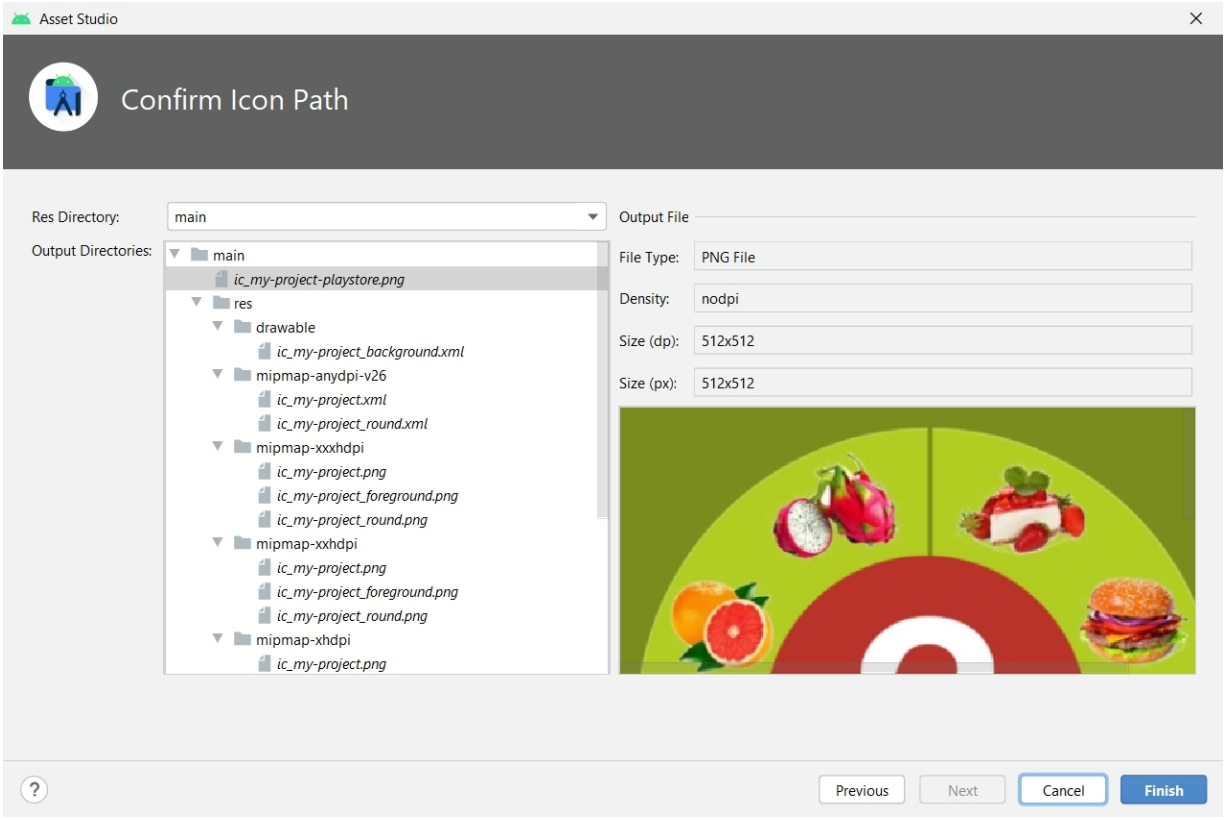
MegaFon #1
77 %
11:37
Добро пожаловать!
Введите своё имя
Начать

Шпаргалка № 26. Меняем иконку приложения.

У любого приложения должна быть своя индивидуальная иконка. Чтобы её изменить приготовьте файл формата .jpg и размером 512×512. Нажимаем File → New → Image Asset. В строке Path выбираем путь к файлу, в поле Name вводим имя, нажимаем Next.



В появившемся окне нажимаем Finish.



Мы создали иконку в проекте, но для того, чтобы её применить, переходим в файл AndroidManifest.xml и меняем две строки:

android:icon="@mipmap/ic_launcher"

android:roundIcon="@mipmap/ic_launcher_round"

На строки:

android:icon="@mipmap/ic_my_project"

android:roundIcon="@mipmap/ic_my_project_round"

Теперь при загрузке приложения в телефоне будет отображаться именно ваша иконка.

Шпаргалка № 27. Имя приложения.

За отображаемое в телефоне имя приложения отвечает строка в манифесте:

android:label="@string/app_name"

Для того, чтобы задать собственное имя приложения, поменяйте его название по умолчанию в строковых ресурсах на своё, какое захотите, например:

<**string name="app_name"**>Калькулятор</**string**>
<**string name="app_name"**>Поиск сокровищ</**string**>
<**string name="app_name"**>Справочник лекарств</**string**>

Шпаргалка № 28. Ориентация приложения.

Например, если Вы хотите, чтобы при использовании приложения оно не поворачивалось при наклоне телефона, а всегда было в вертикальном положении, укажите это в файле манифеста в строке:

android:screenOrientation="portrait"

Если же приложение должно быть всегда в альбомной ориентации, то ставим:

android:screenOrientation="landscape"

Таким образом, в книжном варианте приложение работать не будет.

Шпаргалка № 29. Build.Gradle.

Перед сборкой приложения обязательно загляните в файл сборщика build.gradle текущего модуля. Внимательно посмотрите на параметры, при необходимости подкорректируйте их. Если какие-то библиотеки уже устарели, система подскажет Вам об этом.

Также здесь прописываются такие параметры, как минимальная и целевая версия SDK, номер версии и код сборки, это нужно для того, чтобы в дальнейшем обновлённая версия отличалась от предыдущей. Данная страница выглядит так:

```
plugins {
    id 'com.android.application'}
android {
    compileSdkVersion 31
    buildToolsVersion "30.0.3"
    defaultConfig {
        applicationId "com.kta.mytestproject"
        minSdkVersion 21
        targetSdkVersion 31
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"}
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'}
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8}
}
dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.2'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Шпаргалка № 30. Сборка приложения.

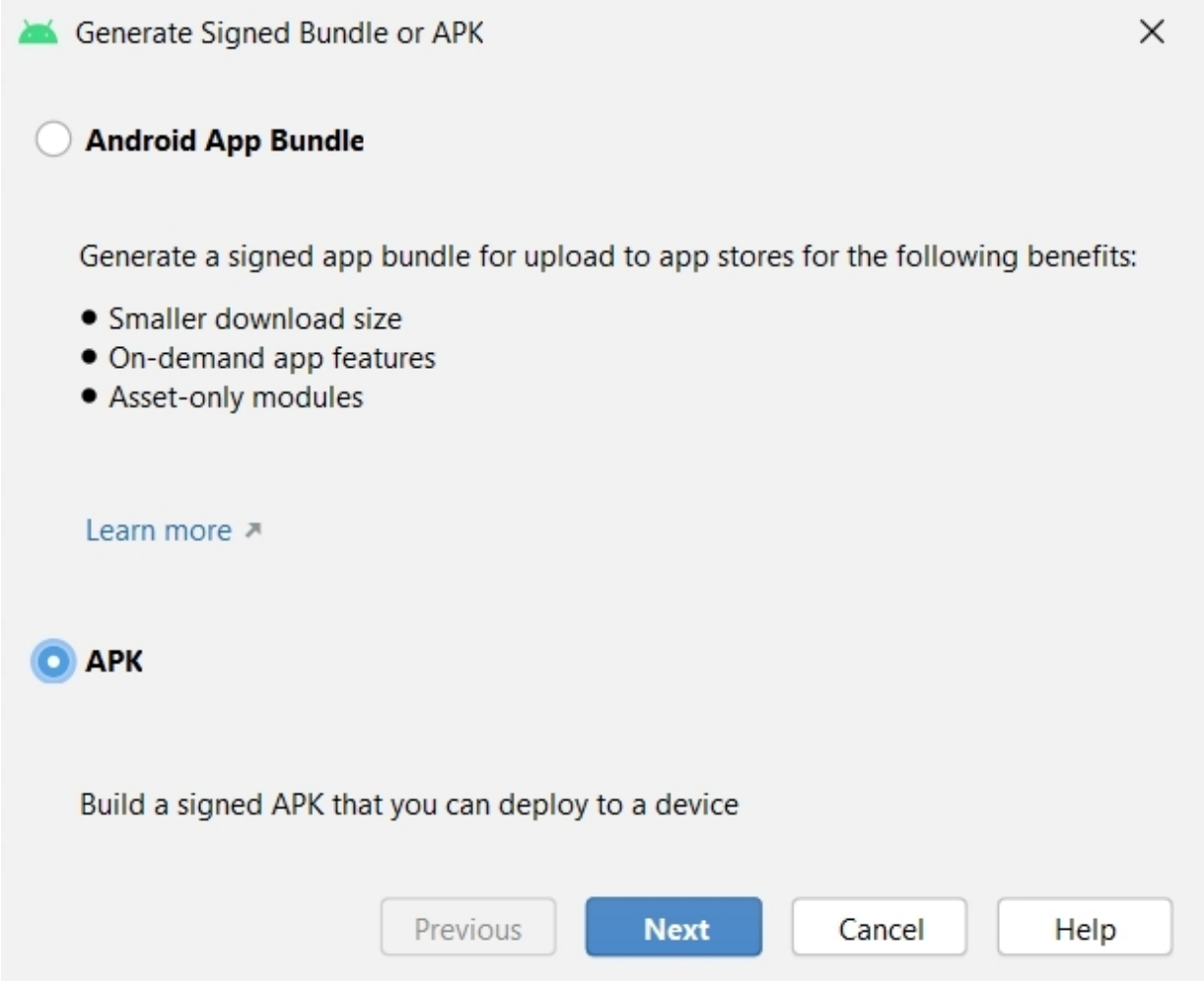
Если у вас всё готово, приложение можно собрать и выложить готовый файл в любой магазин приложений, например PlayMarket, AppGallery, NashStore и другие, а также поделиться своим творением с друзьями и родственниками.

Приложение можно собрать в двух вариантах, apk или aab (набор Bundle). В последнее время PlayMarket принимает приложения только в таком виде, остальные магазины только в формате apk.

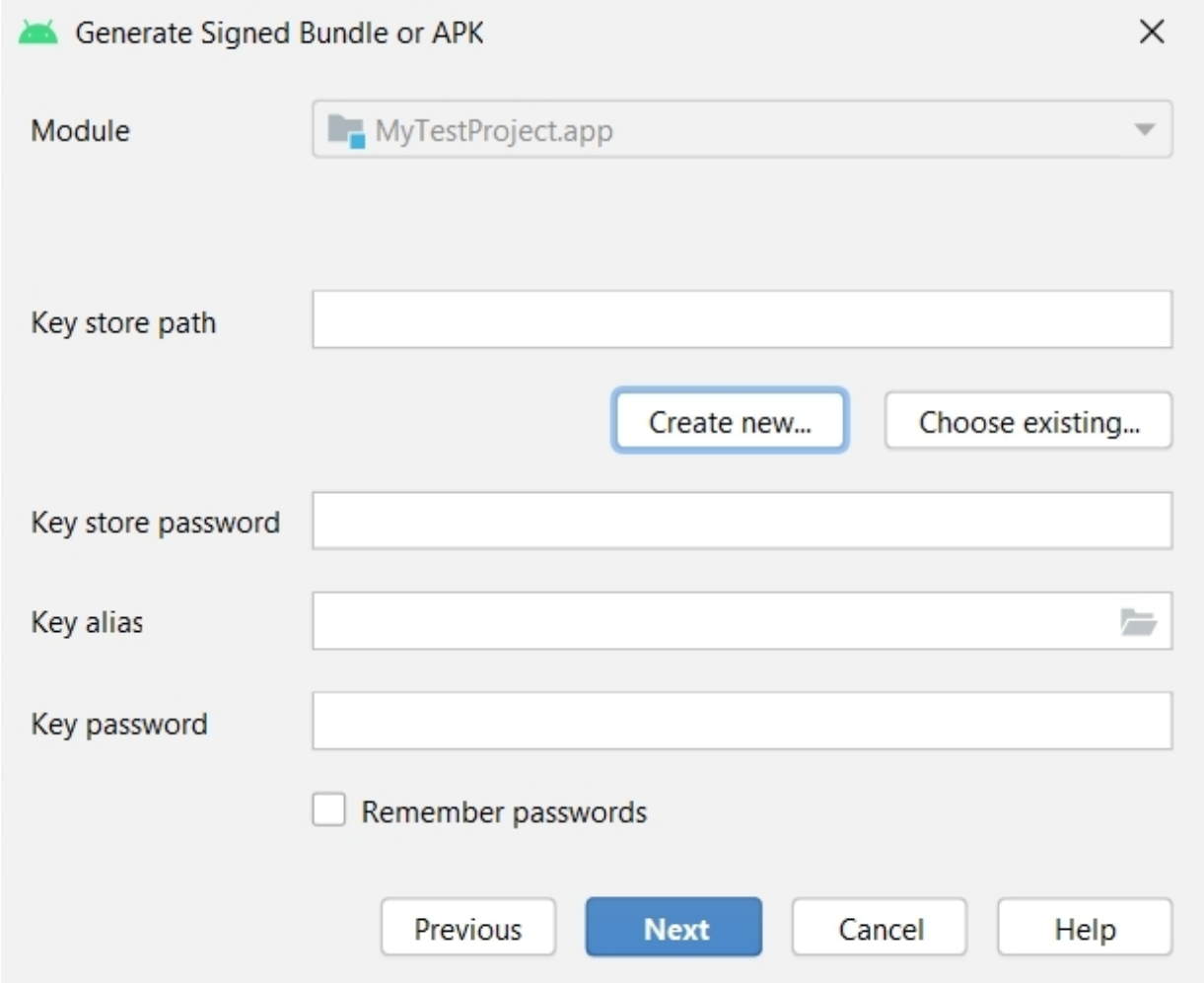
Выбираем пункт Build → Generate Signed Bundle / APK, далее указываем желаемый формат готового приложения.

Если вы хотите поделиться своим приложением с друзьями не через магазины приложений, то выбирайте только APK.

Нажимаем кнопку Next.

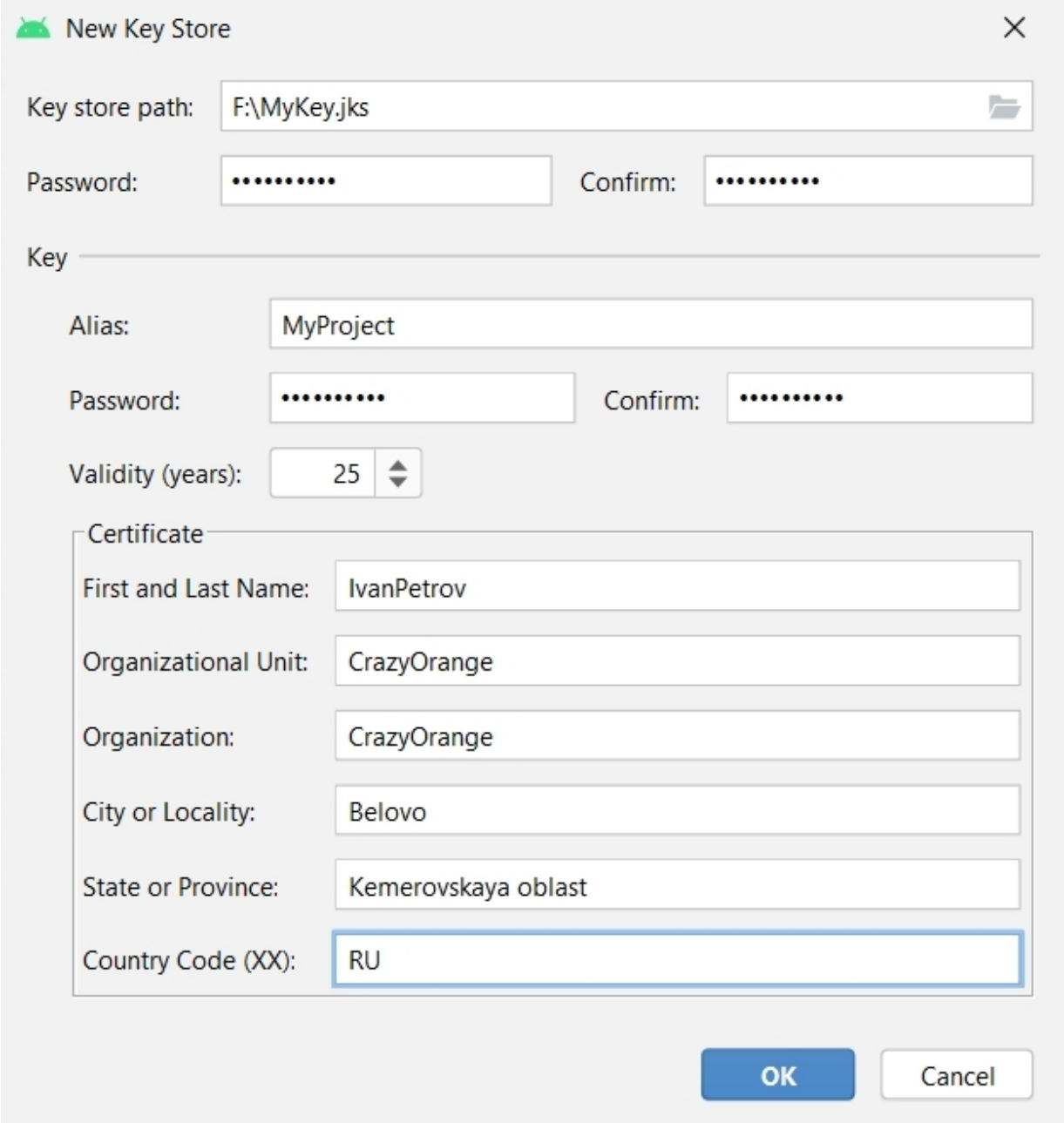


Все приложения для маркетов обязательно нужно подписать специальным ключом. Для этого его нужно создать, нажимаем Create new…

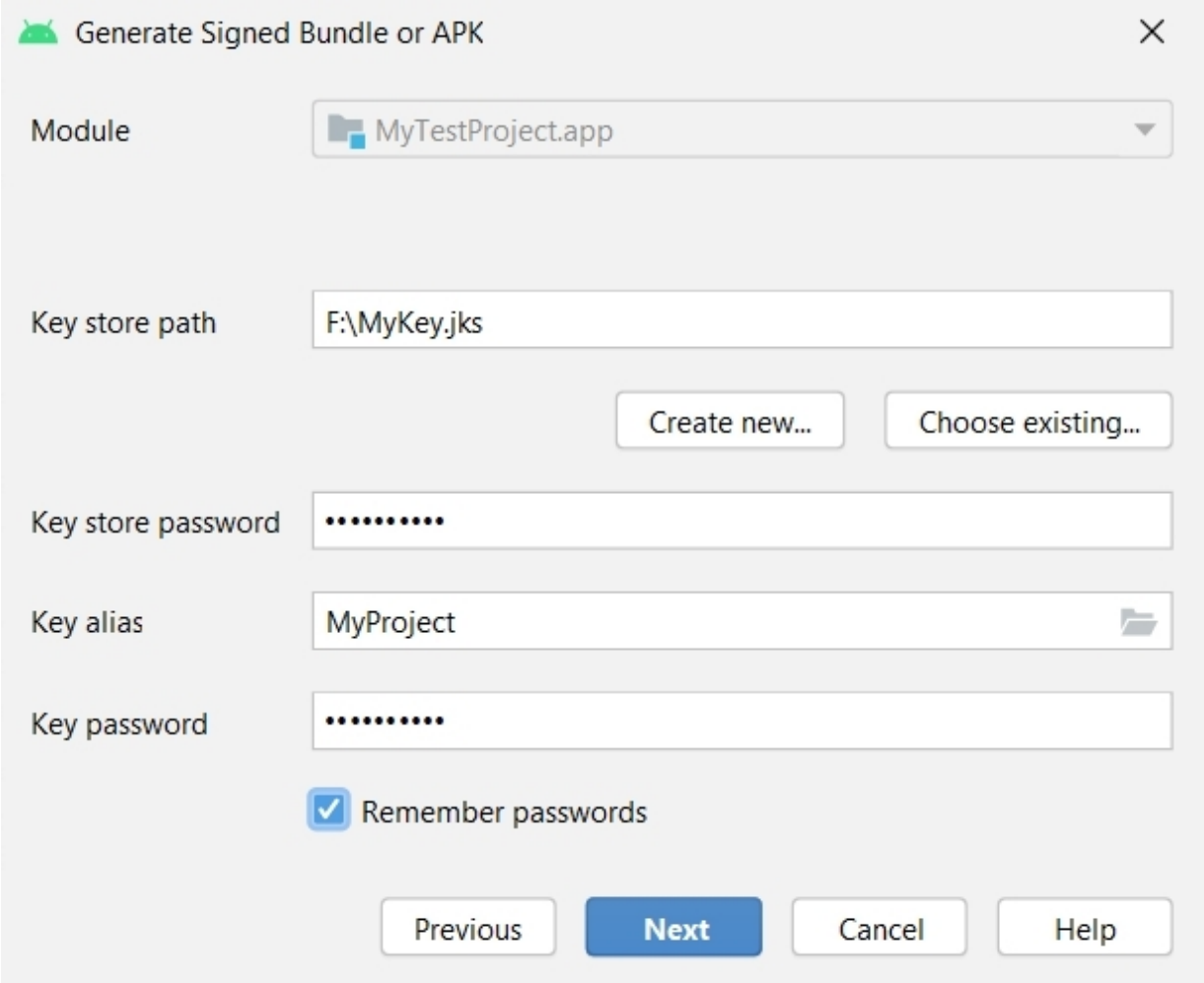


В появившемся окошке заполняем свои данные, выбираем место расположения хранения ключа, придумываем пароли (а лучше записываем, чтобы не забыть).

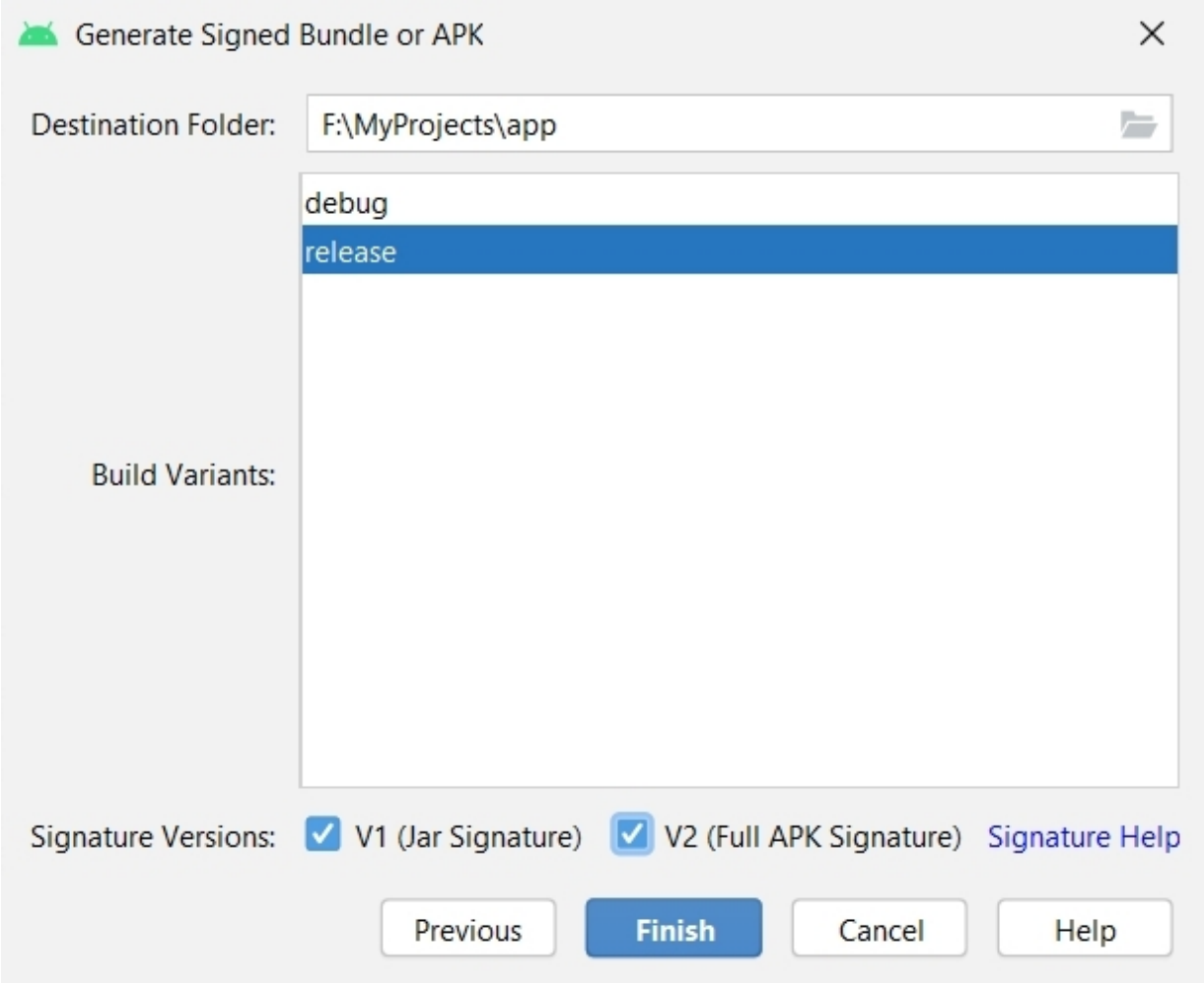
Нажимаем ОК.



Все данные автоматически подставляются. Нажимаем Next.



В следующем окне мы видим, куда сохранится готовое приложение, выбираем release и ставим галочки на V1 и V2. Нажимаем Finish.

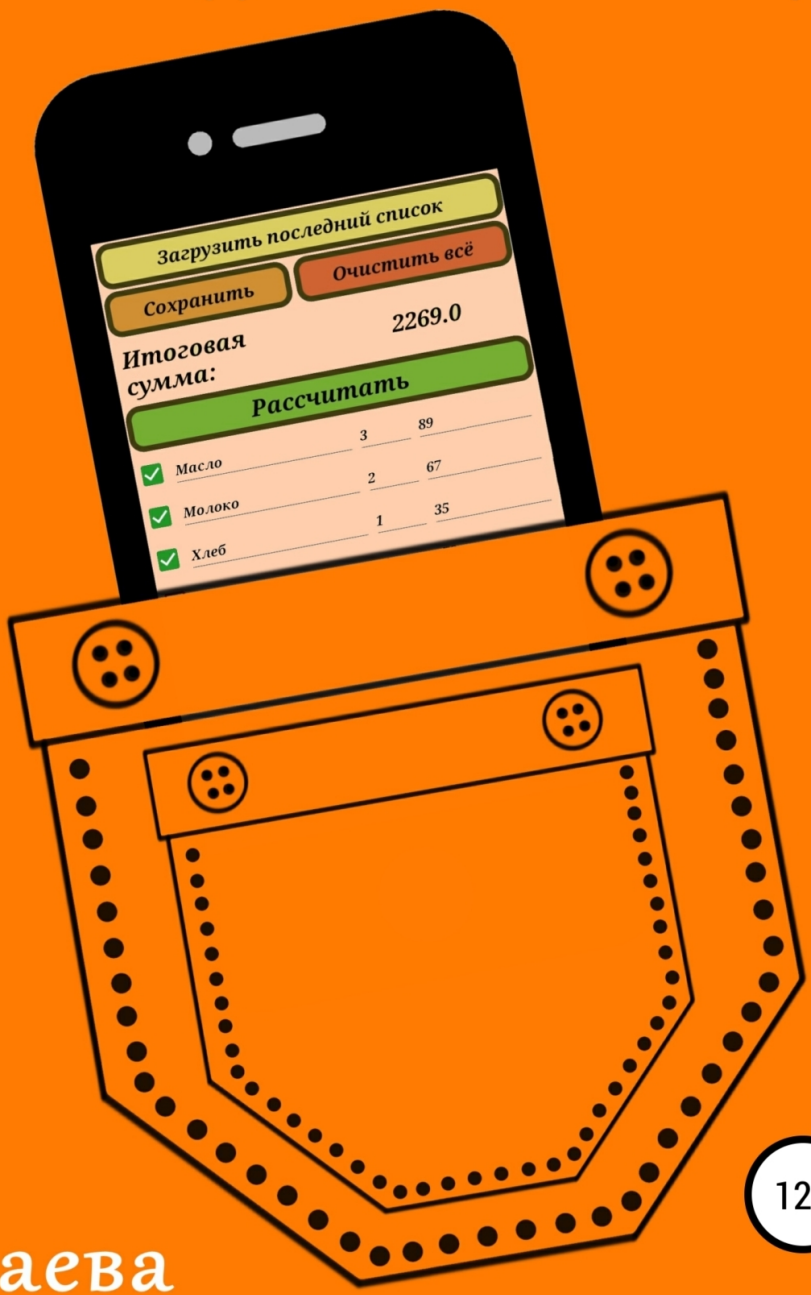


Ждём несколько минут, пока приложение проходит сборку и находим готовый файл APK в папке приложения.

Теперь мы можем установить его на свой телефон, поделиться им с близкими или выложить в магазине приложений для скачивания.

Android Studio

полезные шпаргалки для начинающих



12+

Т. Кучаева